Using CSC Environment Efficiently

September 13th, 2016

Lecturers:

Tomasz Malkiewicz

Atte Sillanpää

Program

09:00 - 09:15 Introduction to the course
09:15 – 09:45 Scientist's User Interface (SUI): an introduction to web-based access to CSC's services
09:45 – 10:00 Coffee break
10:00 - 11:00 How to connect: how to access CSC's computers, NX client, taito-shell demo
11:00 - 12:00 CSC's computing environment: different platforms, module system, licensing, storage and data transfer
12:00 - 13:00 Lunch break
13:00 - 14:30 Running your jobs, resource-management (a.k.a. batch job) systems
14:30 - 14:45 Coffee break
14:45 - 15:30 Compiling your program (writing makefile, linking, debugging)
15:30 - 15:45 Science services at CSC: a short introduction
15:45 - 16:15 Troubleshooter + Installation session: helping with installation of NX client, PuTTy, Virtual appliance,...

Practicalities

- Keep the name tag visible
- Lunch is served in the same building
- Toilets are in the lobby
- Network:
  - WIFI: eduroam, HAKA authentication
  - Ethernet cables on the tables
  - CSC-Guest accounts
- Bus stops
  - Other side of the street (102,103) → Kamppi/Center
  - Same side, towards the bridge (194,195/551) → Center/Pasila
  - Bus stops to arrive at CSC at the same positions, just on opposite sides
- *If you came by car: parking is being monitored - ask for a temporary parking permit from the reception (tell which workshop you're participating)*
- Visiting outside: doors by the reception desks are open
- Room locked during lunch
  - Lobby open, use lockers
- Username and password for *workstations*: given on-site

CSC at a Glance

CSC?

- Non-profit company owned by Ministry of Education and Culture
- Services mainly free for researchers
- In 2015: About 2700 active users
- Applications, computational capacity, user support, FUNET, information management services, data services
- Participating in 15 EU projects

Internationally competitive research environments and e-Infrastructures

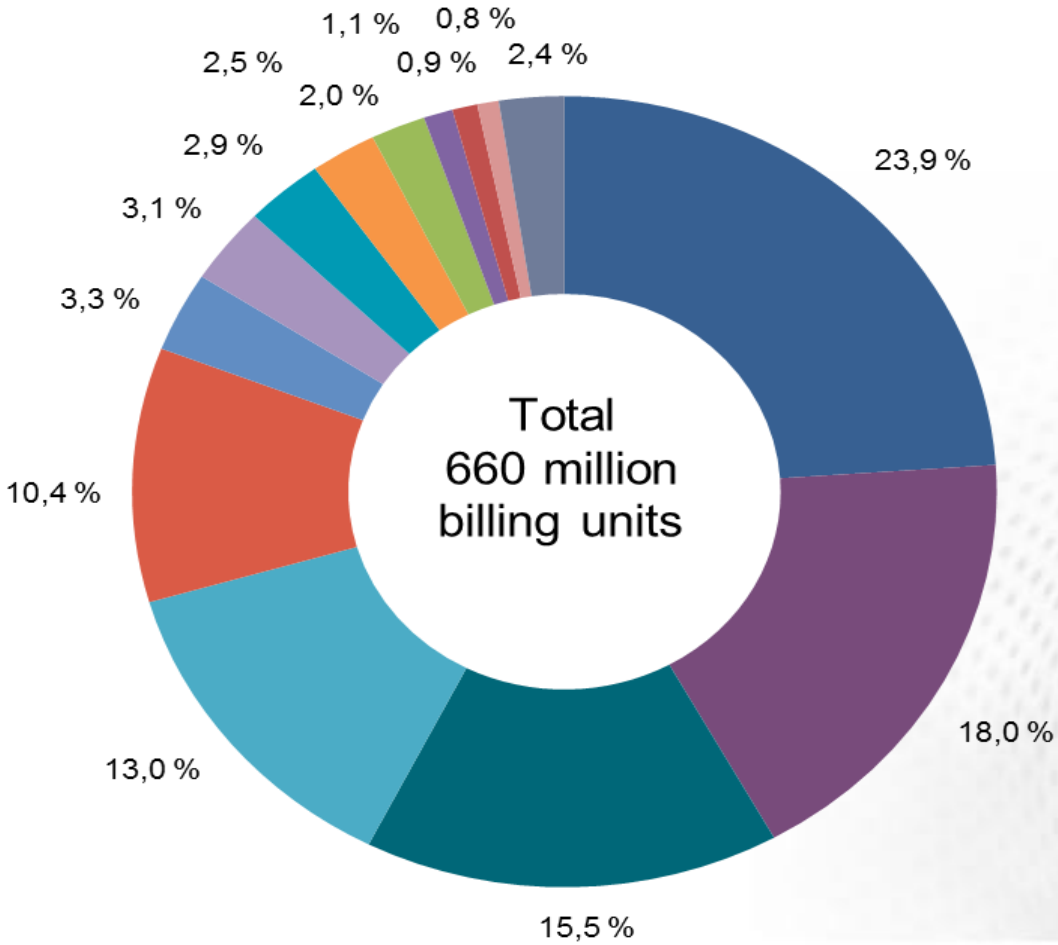
Collaboration with majority of European computing centers

- International research network organizations:
  - *NORDUnet, TERENA, GÉANT (GN3)*
- European research infrastructures and supporting projects:
  - *ELIXIR, CLARIN, ENVRI*
- International HPC projects and GRID-organizations:
  - *Nordic e-Infrastructure Collaboration (NeIC), PRACE, EGI-Inspire*
- European e-Infrastructure policy initiatives :
  - e-Infrastructure Reflection Group (e-IRG), RDA

Datacenter CSC Kajaani

- CSC's modular Data Center in Kajaani. Modern and reliable infrastructure (national power grid, roads, airline connections, data networks)
- The Funet Network ensures excellent networking capabilities around the world
- Place for CSC's next supercomputers with other CSC customer systems
- Cost-Efficient Solution – Sustainable and Green Energy Supply

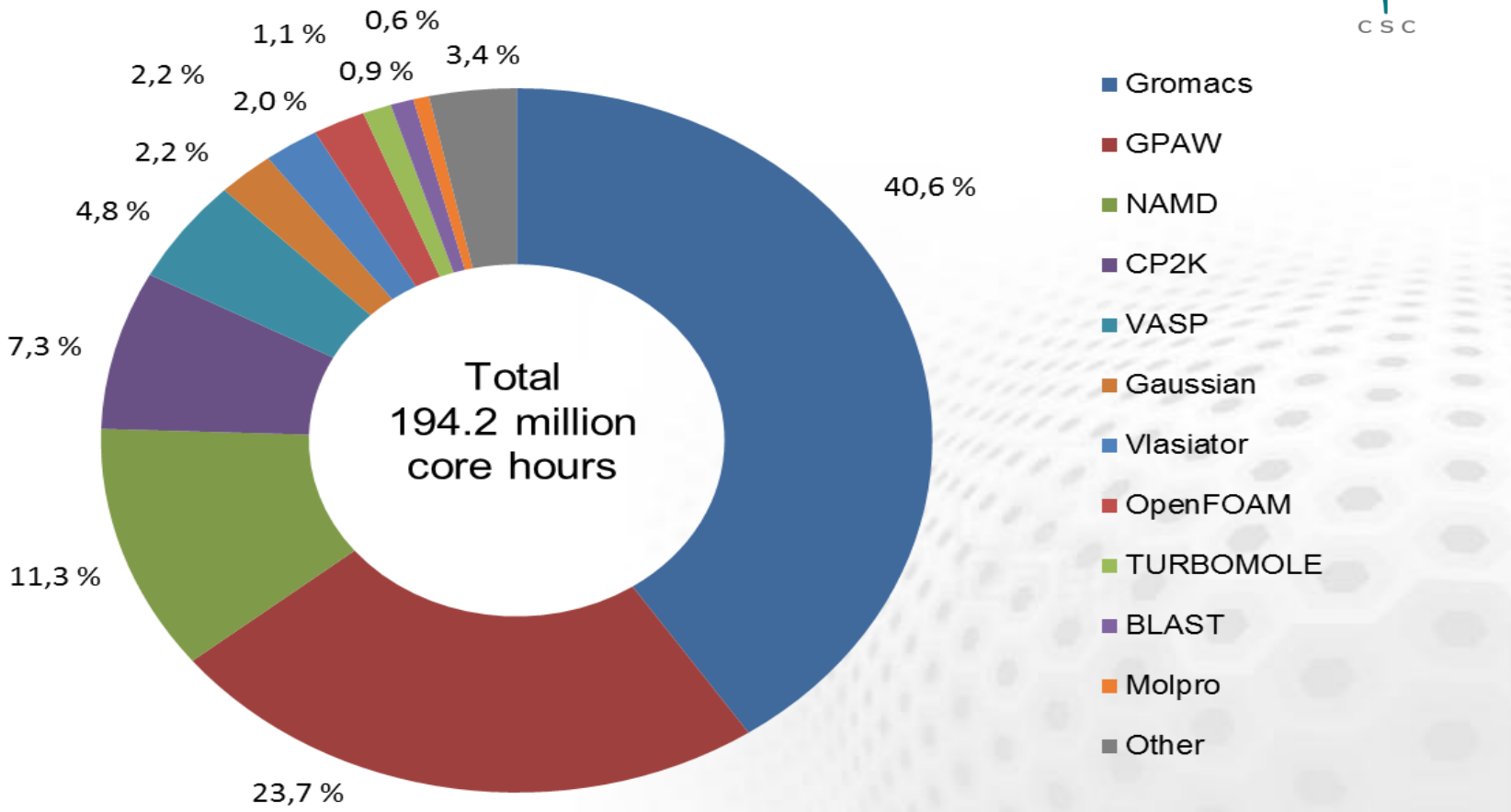
Computing usage by organization 2015



- Tampere University of Technology
- University of Helsinki
- Aalto University
- University of Jyväskylä
- CSC Grand Challenge Projects
- International Usage
- University of Turku
- Lappeenranta Univ. of Technology
- University of Oulu
- CSC Projects
- PRACE (Tier-1)
- Åbo Akademi
- University of Eastern Finland
- Other

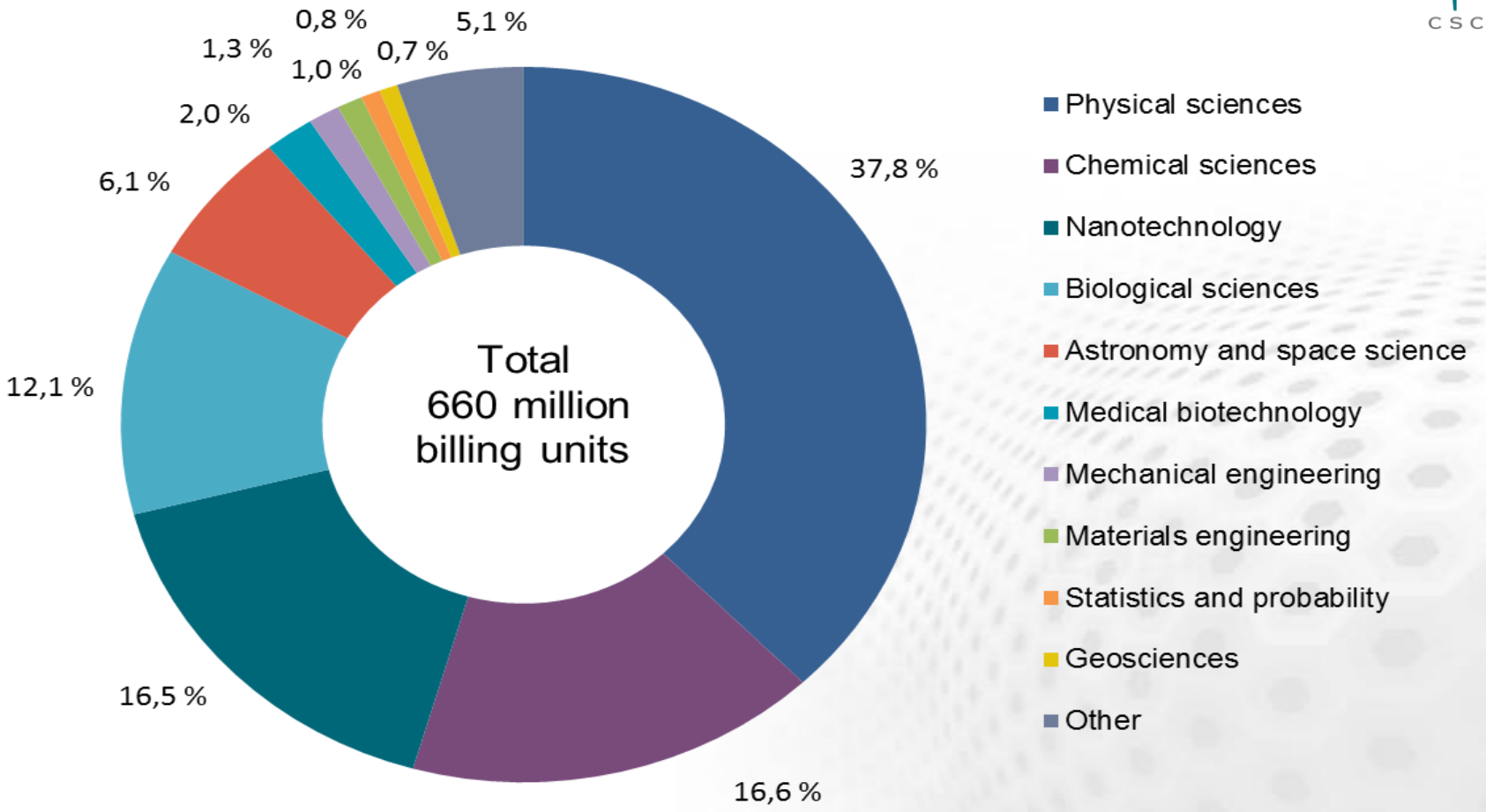
Software usage (maintained by CSC) 2015

(Usage according to Process Accounting system as core hours)

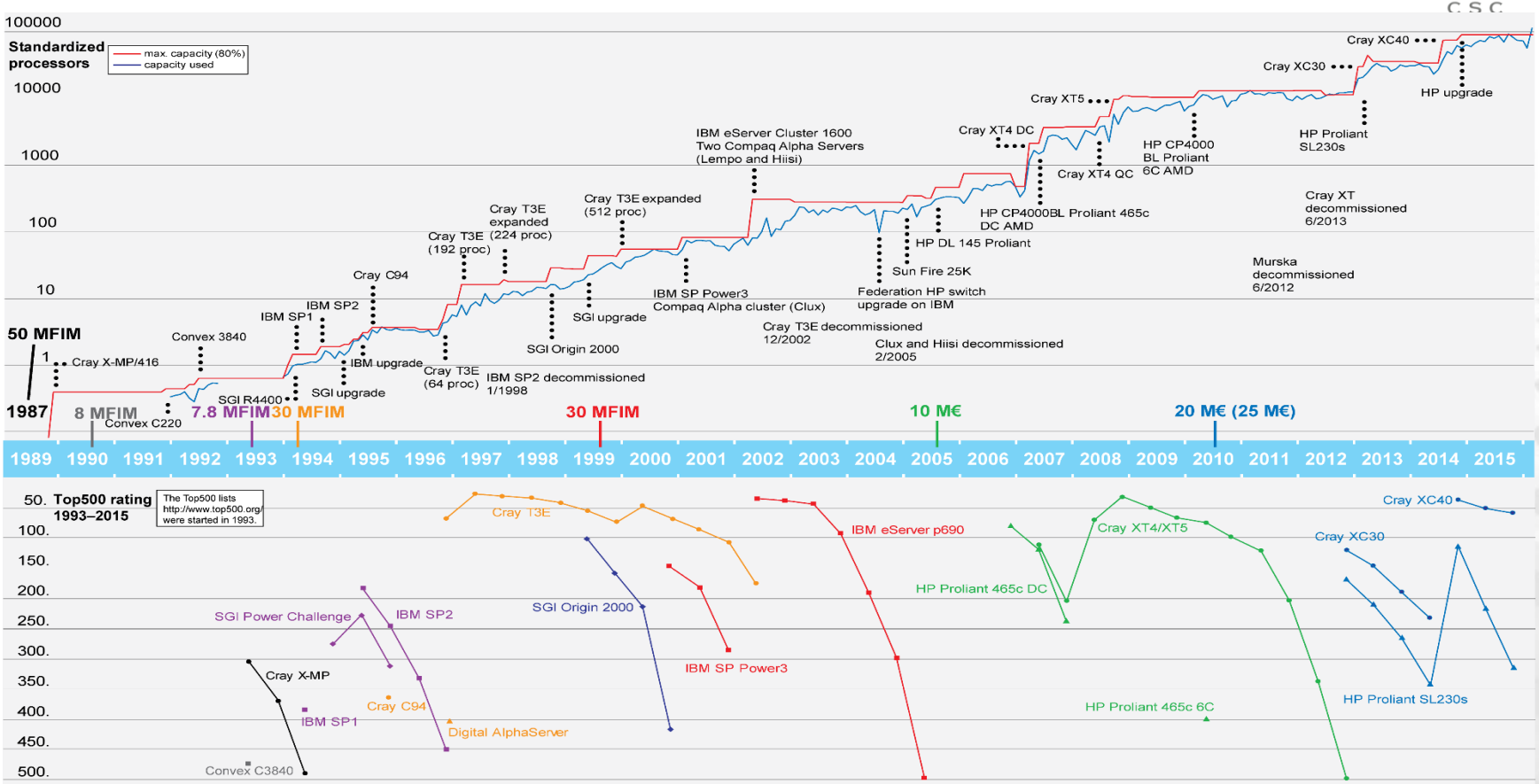


CSC maintained software's usage covers over 60% of all computing time usage

Computing usage by discipline 2015
(includes Sisu, Taito and cPouta usage)

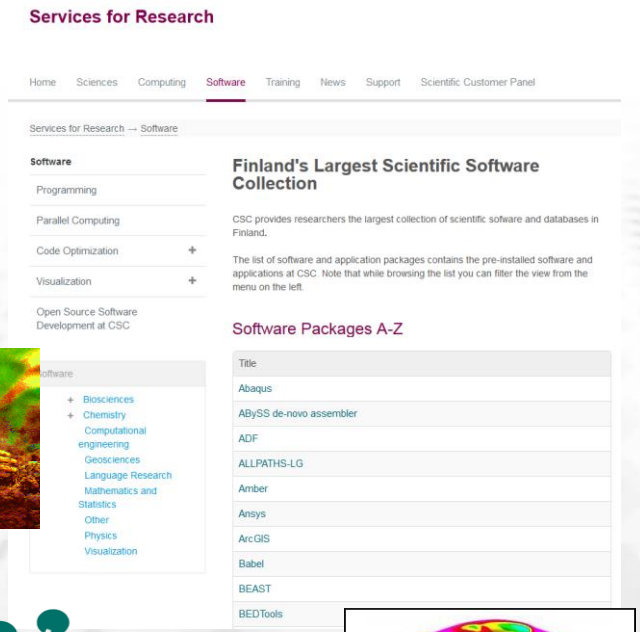


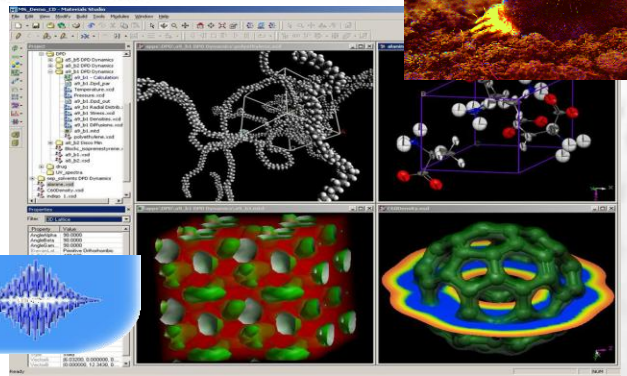
CSC's Computing Capacity 1989–2015



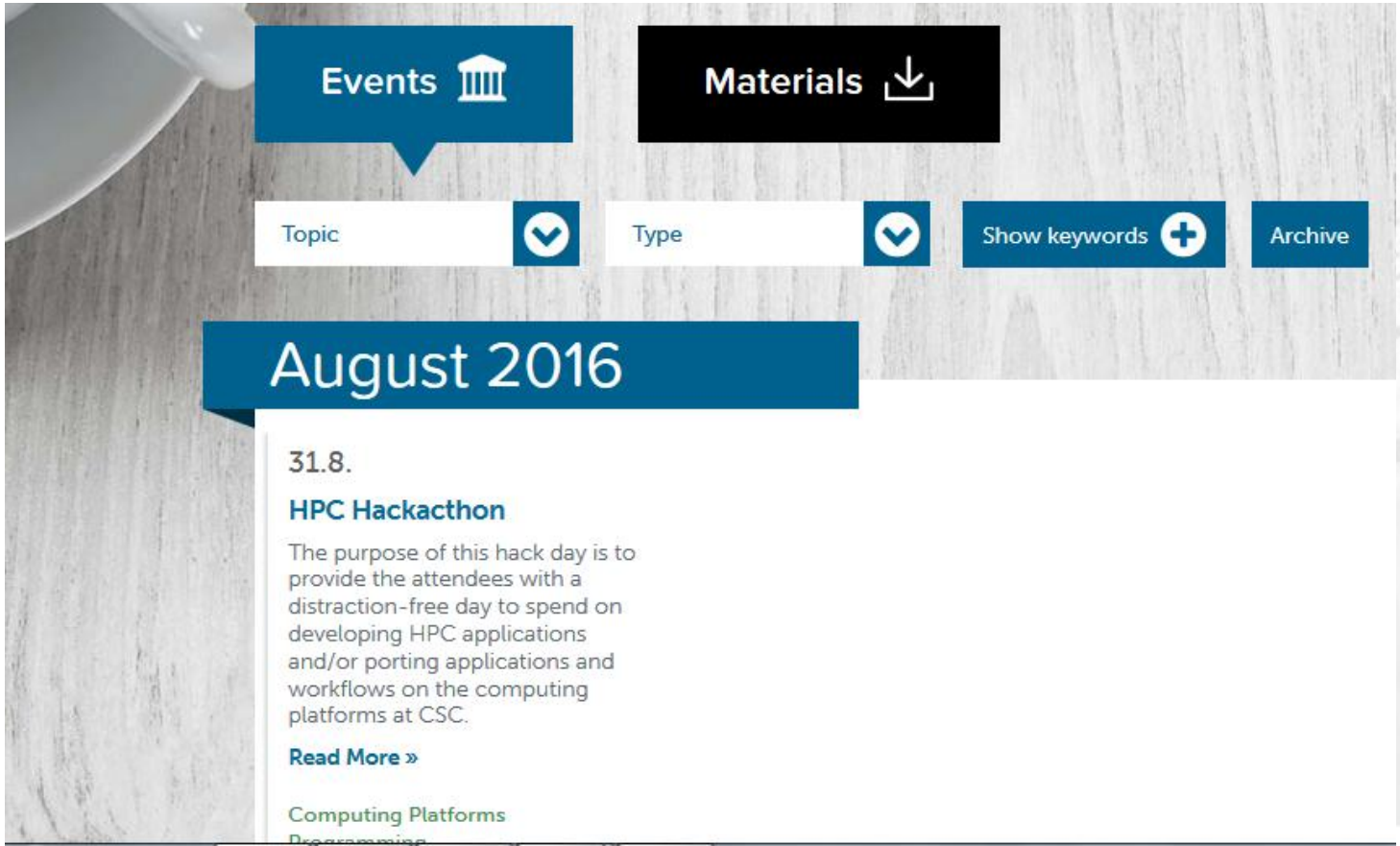
Software and database offered by CSC

- Large selection (over 200) of software and database packages for research https://research.csc.fi/software
- Mainly for academic research in Finland
- Centralized national offering: software consortia, better licence prices, continuity, maintenance, training and support



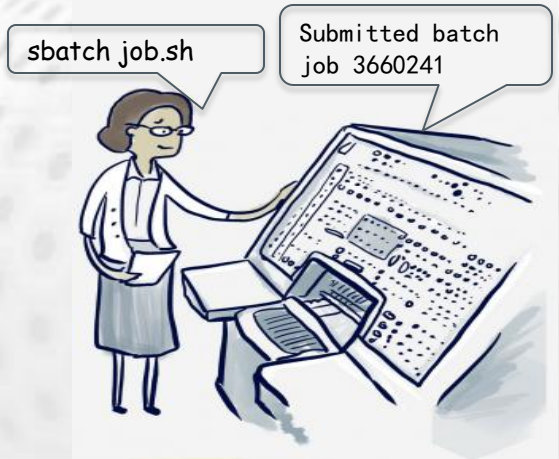


Courses


Events
Materials
Topic
Type
Show keywords
Archive
August 2016
31.8.
HPC Hackacthon
The purpose of this hack day is to provide the attendees with a distraction-free day to spend on developing HPC applications and/or porting applications and workflows on the computing platforms at CSC.
Read More »
Computing Platforms

How to get started?

- https://research.csc.fi
- https://research.csc.fi/csc-guide
- https://research.csc.fi/faq-knowledge-base
- https://www.csc.fi/web/training/materials → CSC-Environment
- Service Desk: servicedesk@csc.fi



Getting access to CSC resources

1. Register: User account

- https://research.csc.fi/csc-guide-getting-access-to-csc-services
- Login via HAKA autentication to SUI https://sui.csc.fi
  - There you find the Registration functionality "*Sign Up*"
- This will get you an initial computing quota
  - Sending computation job consumes processor cores
  - User gets a Personal Project with 10'000 billing units (5000 core-hours) and access to Taito cluster.
    - It is just for piloting, not for large jobs and you cannot apply for addiotional computing quota or services

2. Academic Project

- https://research.csc.fi/csc-guide-projects-and-resource-allocation
- Professors and PIs can apply for **an Academic Project**.
  - Login via HAKA autentication to SUI https://sui.csc.fi
  - Fill the application form for the Academic project
- In SUI *My Projects* tool you can select which project is the Project which quota is used.
  - Thus, change **the default billing project** from your Personal Project to the Academic when you get it!

3. Apply for a Service and billing units

- Only an Academic Project (not a Personal project) can apply access to Service and billing units.
- PI of an Academic Project can apply for **access to Taito, Sisu, cPouta and IDA storage *services*** in SUI
  - https://sui.csc.fi/group/sui/resources-and-applications
- With SUI My Project tool Project Member can apply for **more billing units** for an Academic Project
  - https://sui.csc.fi/group/sui/my-projects

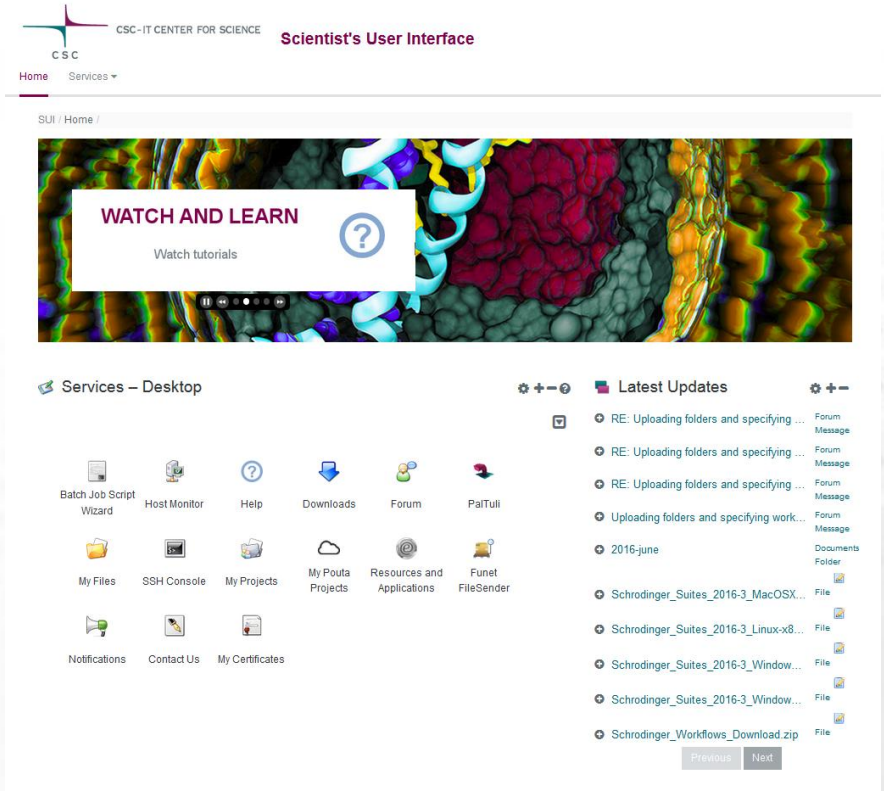
Scientist's User Interface (SUI)

Scientist's User Interface (SUI)

WWW-portal for all CSC users – https://sui.csc.fi

- Sign up as customer
- Reset your password
- Manage your account
- Apply for an Academic project
- Apply for computing services

- Access your data
- Download material
- Watch videos
- Submit jobs
- Monitor hosts and jobs
- Personalize your use
- Message board
- + **more**



Scientist's User Interface (SUI)

Use case – run job via SUI-portal

Generate and store suitable job script with **Batch Job Script Wizard**

Open terminal connection to Taito with **SSH Console** and submit job

or

Submit job with **My Files**

Monitor your job on Taito with **Host Monitor**

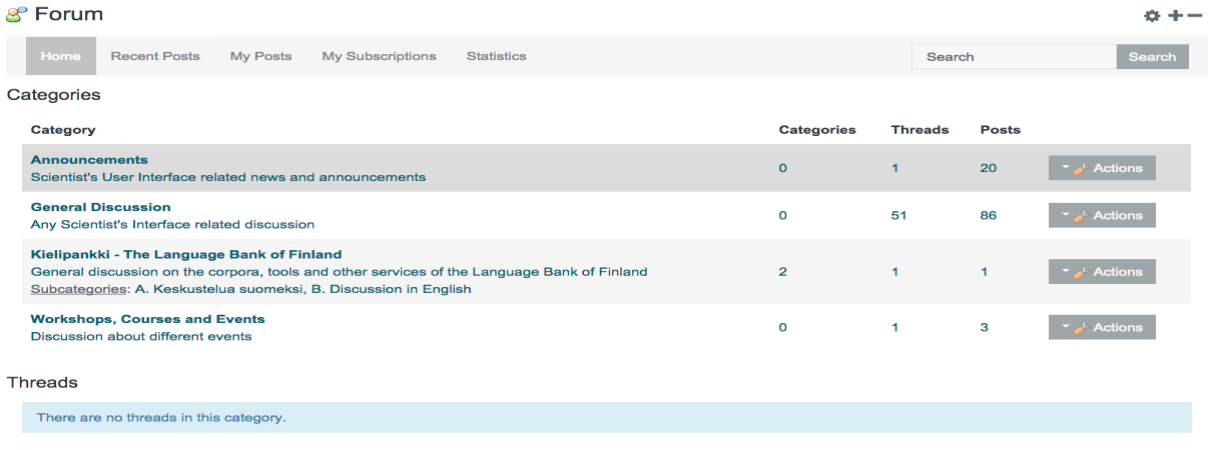
Examine and download results with **My Files**

Monitor your project's resource usage with **My Projects**

Scientist's User Interface (SUI)

Forum

- **Participate** in discussion on forum
- Quick way to find information of SUI, ask questions or **give feedback** to developers
- **Share ideas** for new services



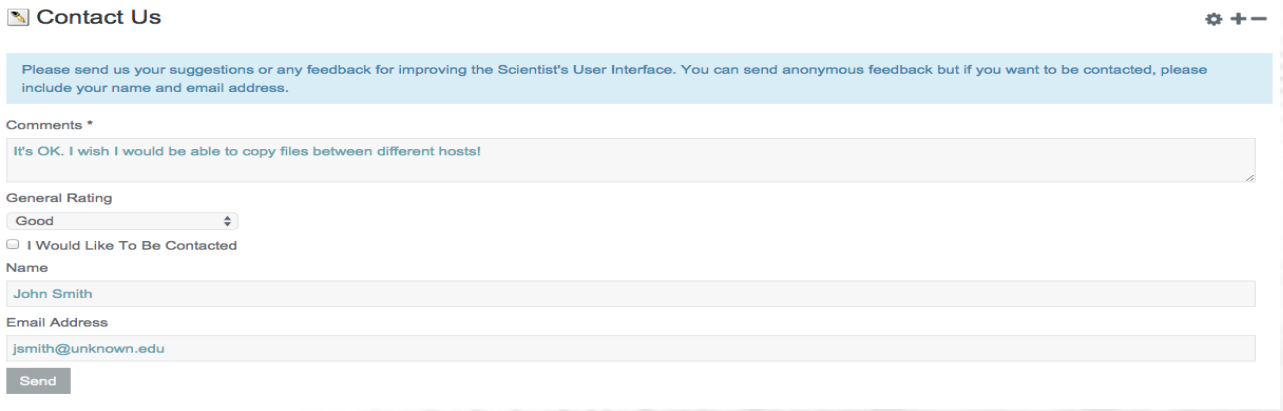
Scientist's User Interface (SUI)

Contact Us

- One way to **contact** or **give feedback**

The main contact: servicedesk@csc.fi

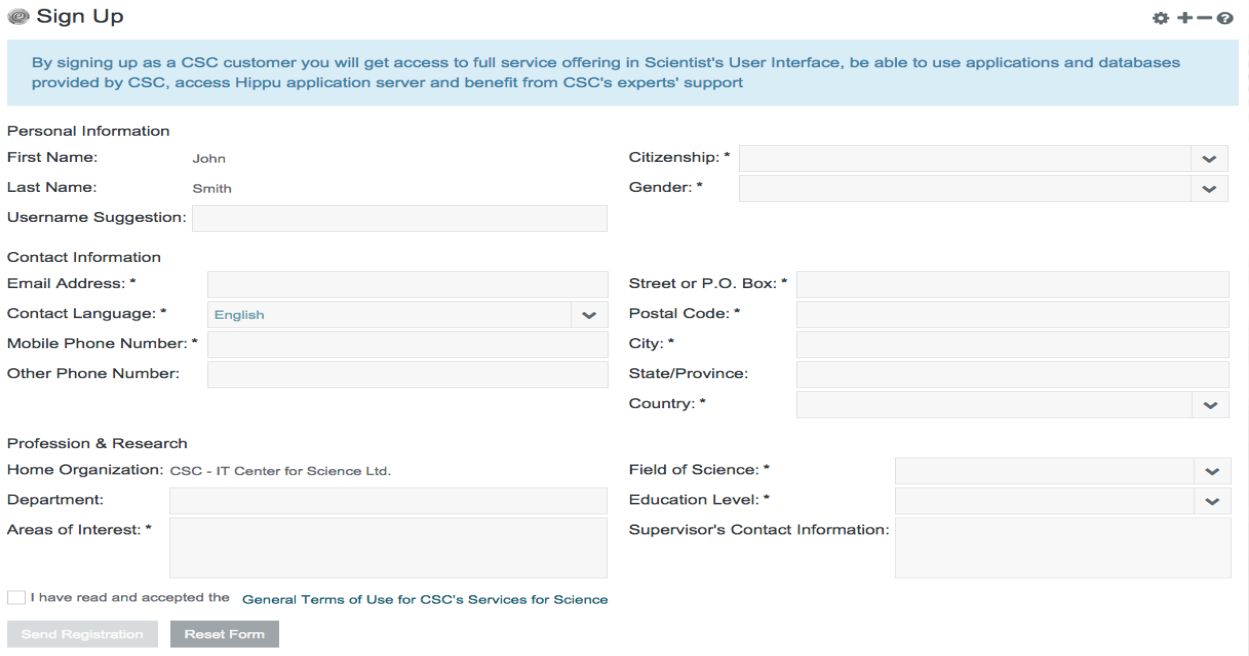
- Direct feedback can be sent privately and anonymously



Scientist's User Interface (SUI)

Sign Up

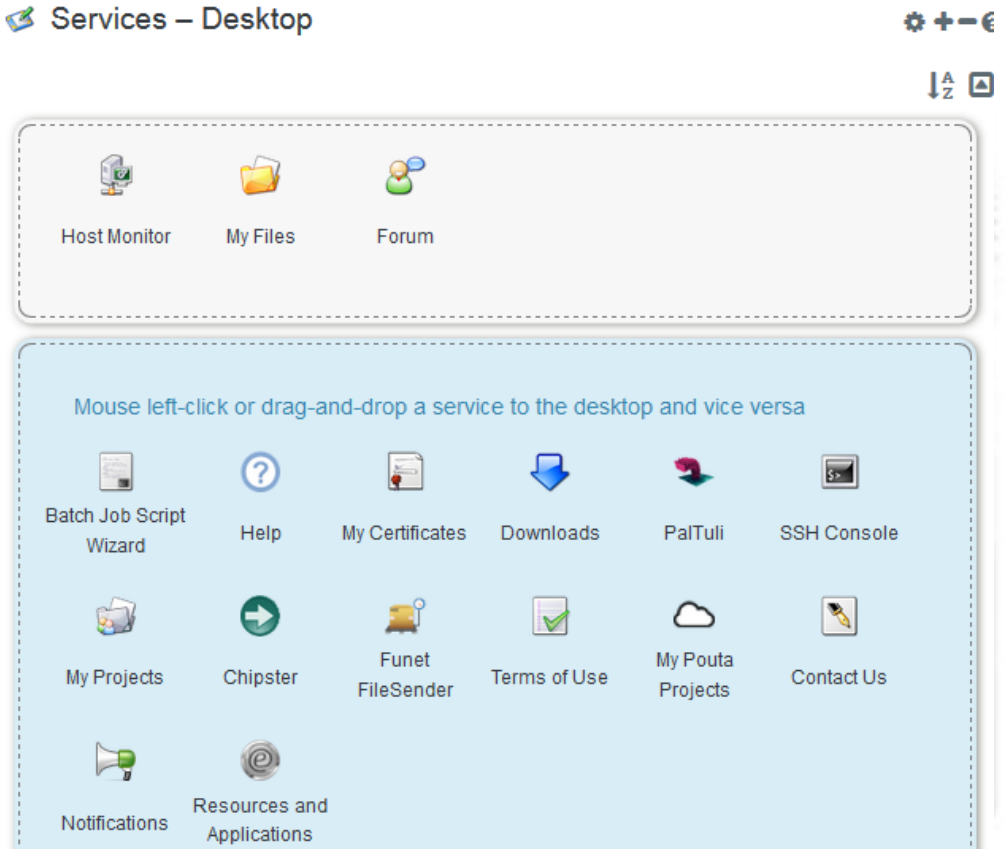
- Quick and easy way to **Sign up** as CSC customer
- Available for all users by **Haka login**
- By signing up you can access all SUI's services, applications and databases, Hippu application server + more



Scientist's User Interface (SUI)

Services - Desktop

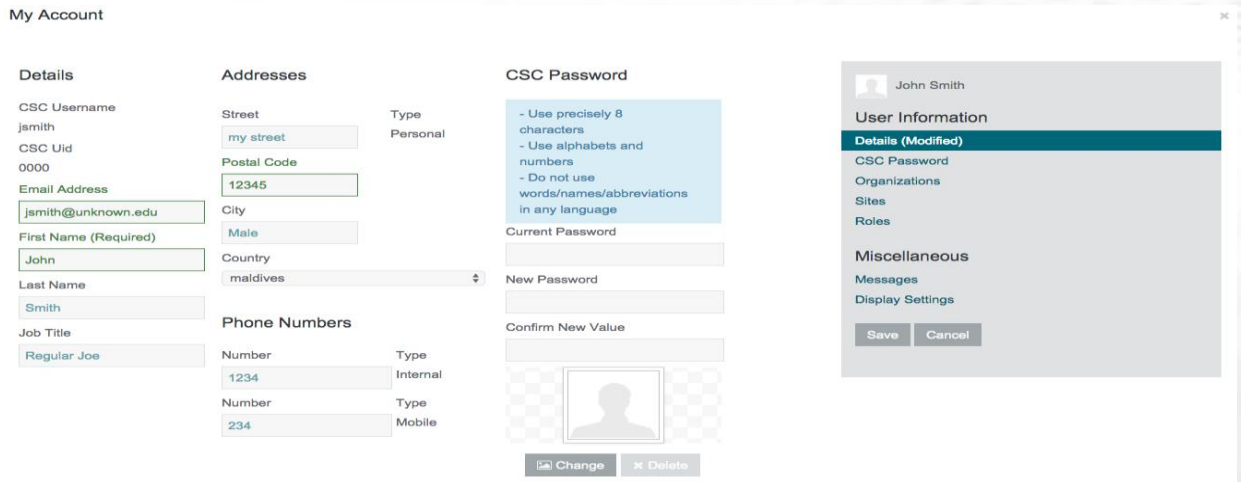
- **Personalize** your desktop by selecting your favorite services
- Sort/arrange by using drag&drop
- See messages



Scientist's User Interface (SUI)

My Account

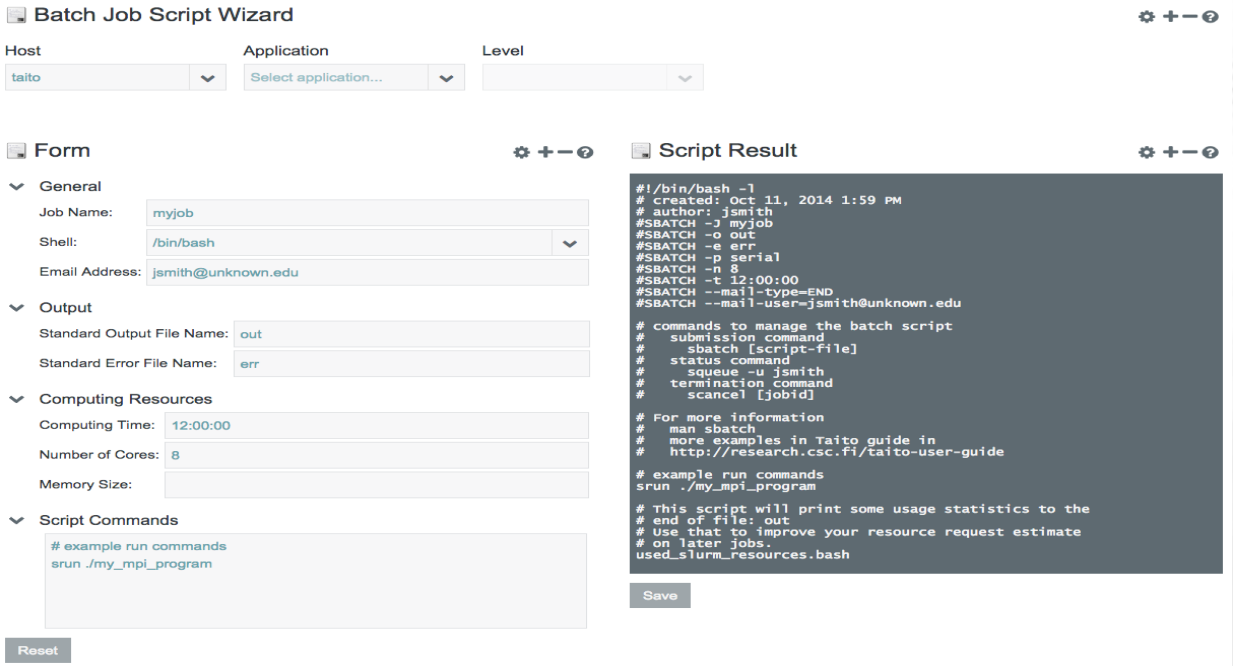
- **Maintain** your account information
- **Change password** for CSC environment
- **Define** your personal settings



Scientist's User Interface (SUI)

Batch Job Script Wizard

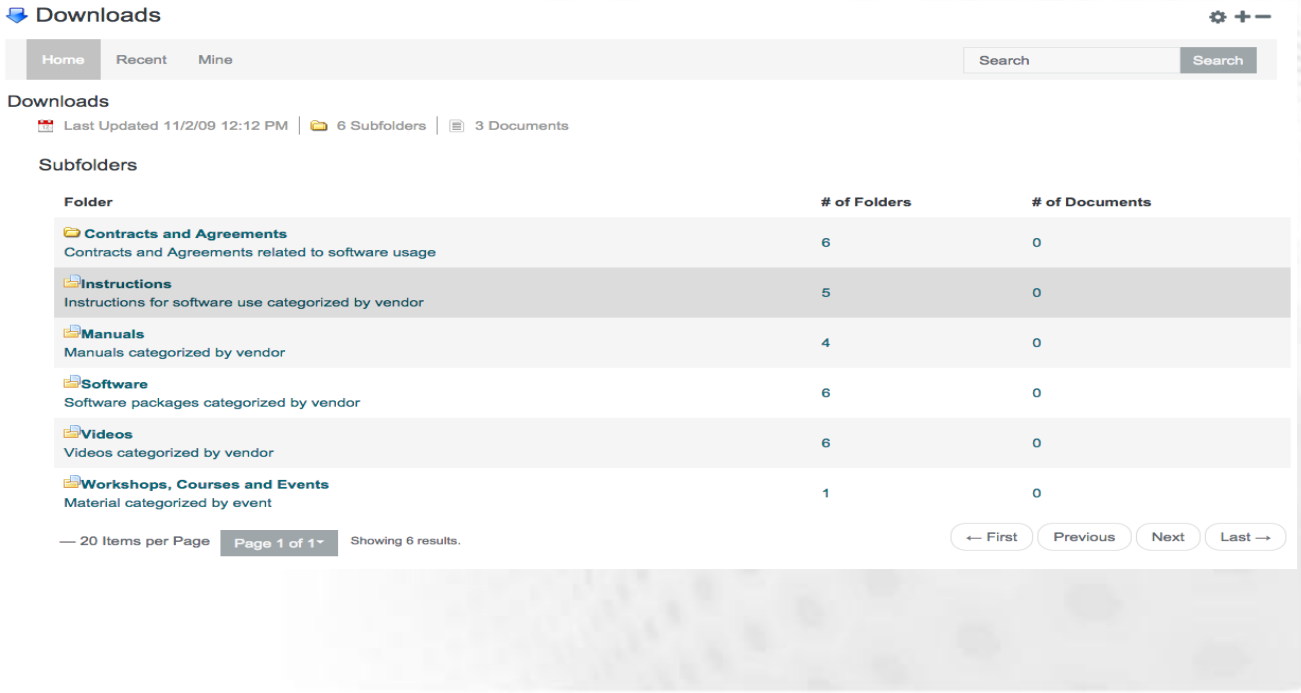
- **Create job scripts** with easy to use forms
- **Save scripts** locally or in CSC $HOME
- Instructions of how to submit and monitor



Scientist's User Interface (SUI)

Downloads

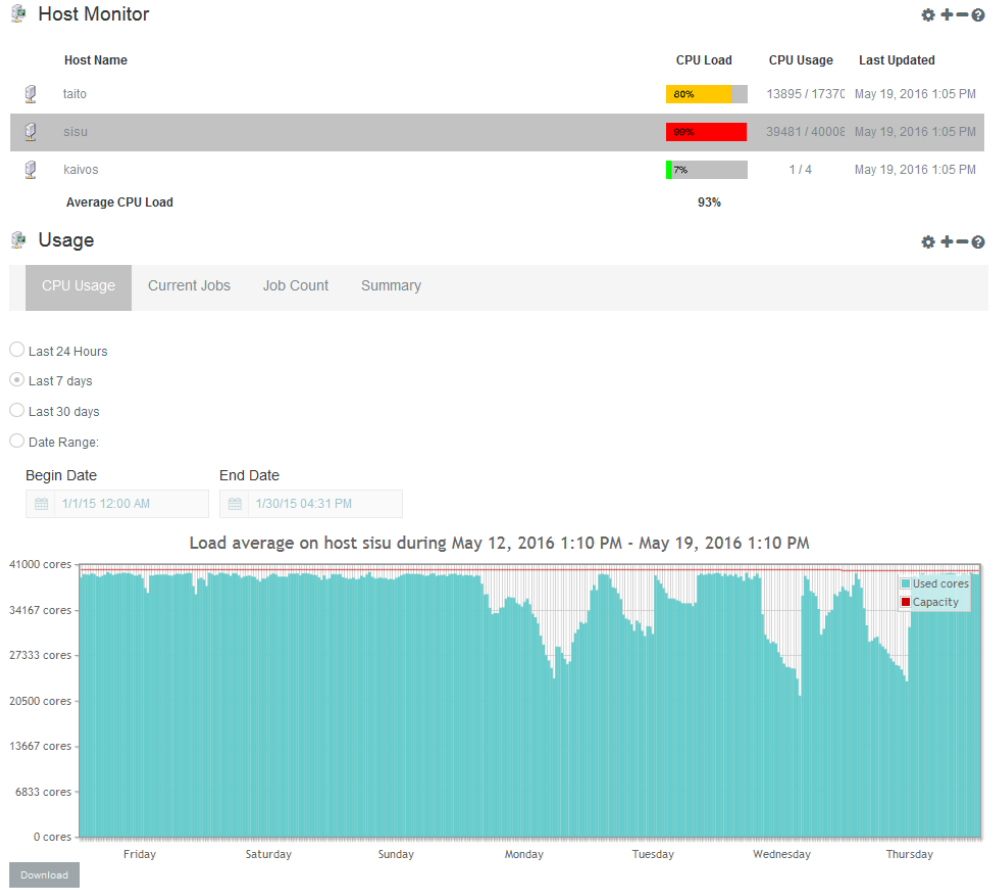
- **Access material** provided to you by CSC
- Software installation packages, manuals, videos etc.



Scientist's User Interface (SUI)

Host Monitor

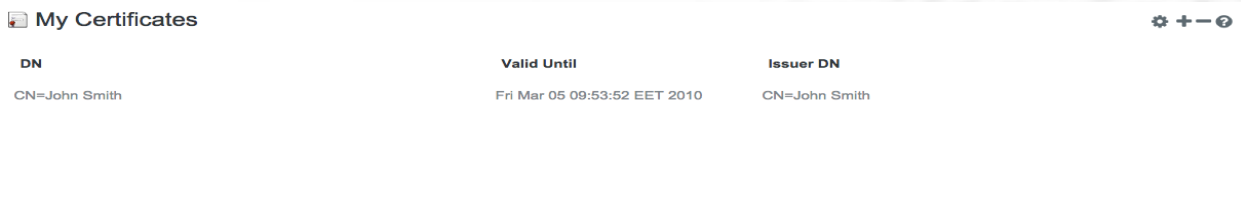
- **View statuses and details** of CSC's computing servers and batch systems
- **Visualize history** of CPU usage and job count
- **Monitor jobs** in all hosts in single view
- **Control** your own jobs



Scientist's User Interface (SUI)

My Certificates

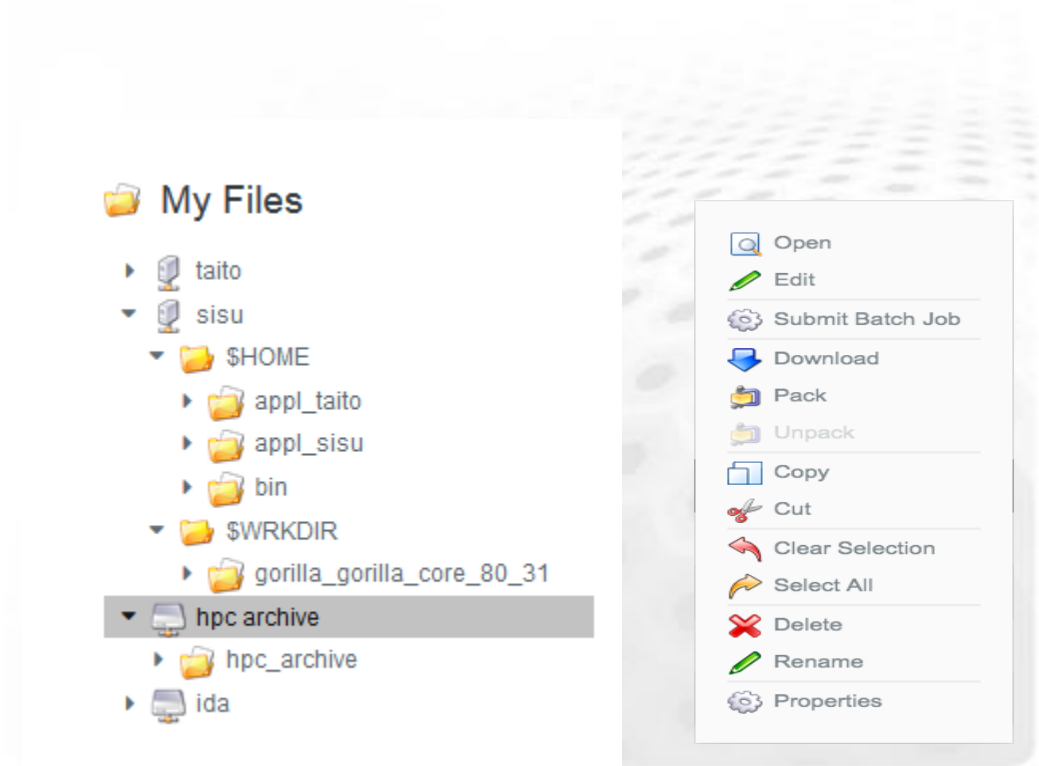
- **Process** your X509 digital certificates
- Format conversions, export proxies, save locally or to your CSC $HOME
- **Setup grid usage** in CSC's computers



Scientist's User Interface (SUI)

My Files

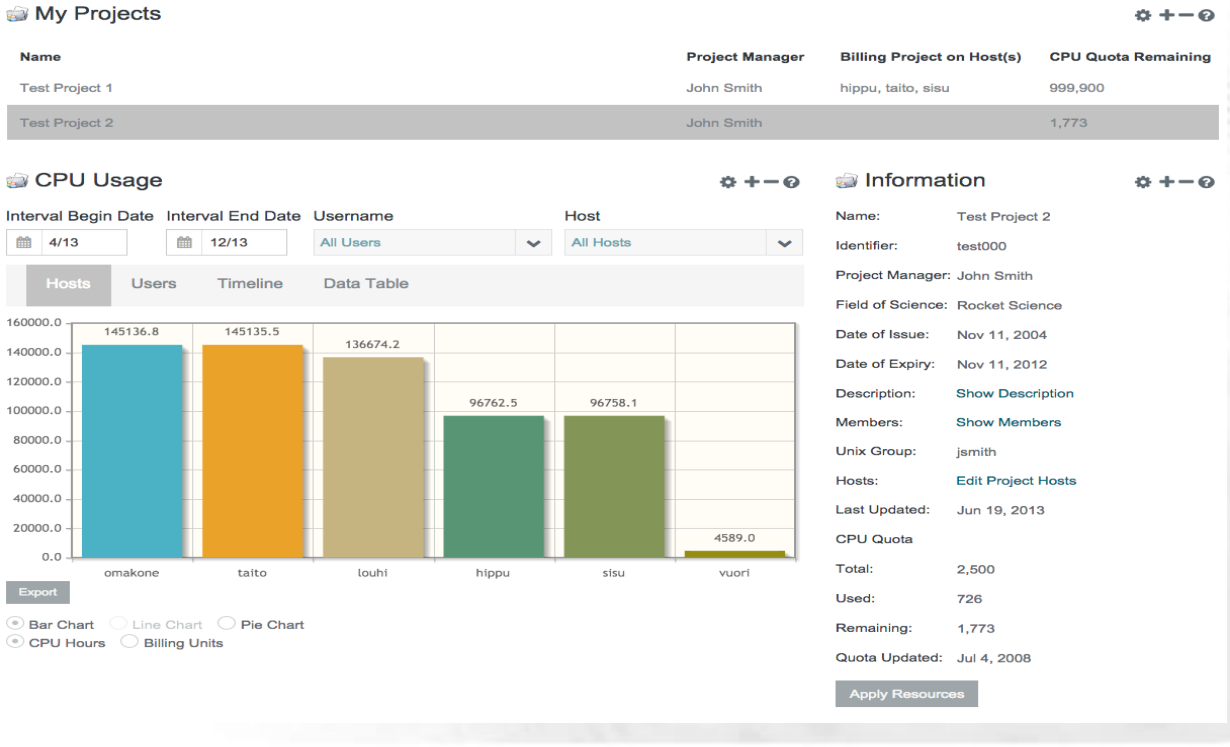
- **Access your data** in CSC's storage services in single view (computing servers, IDA and HPC Archive)
- Transfer files
- **Search** your data
- **Submit** jobs
- Typical folder and file operations are supported



Scientist's User Interface (SUI)

My Projects

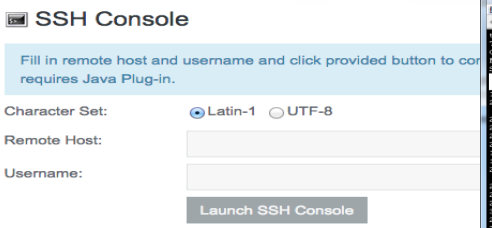
- **View information and resource usage** of your CSC projects
- **Edit hosts** for projects
- **Apply resources** for your CSC customer project
- Resource usage currently not working due system changes

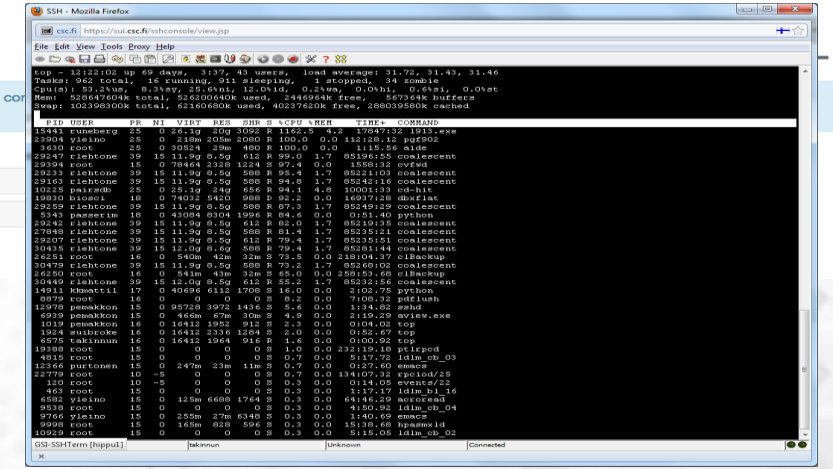


Scientist's User Interface (SUI)

SSH Console

- **Connect** to CSC's computing servers
- UTF-8 character translation support

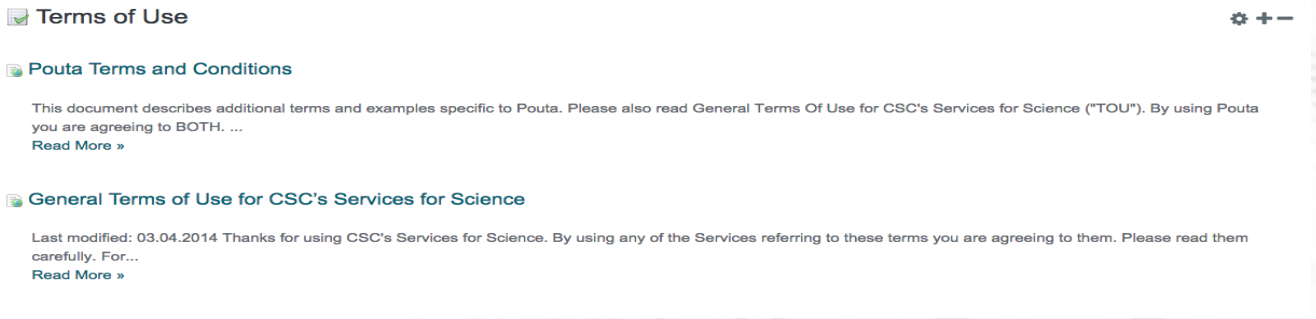




Scientist's User Interface (SUI)

Terms of Use

- **Read** CSC's services' terms of use



Login to SUI via HAKA

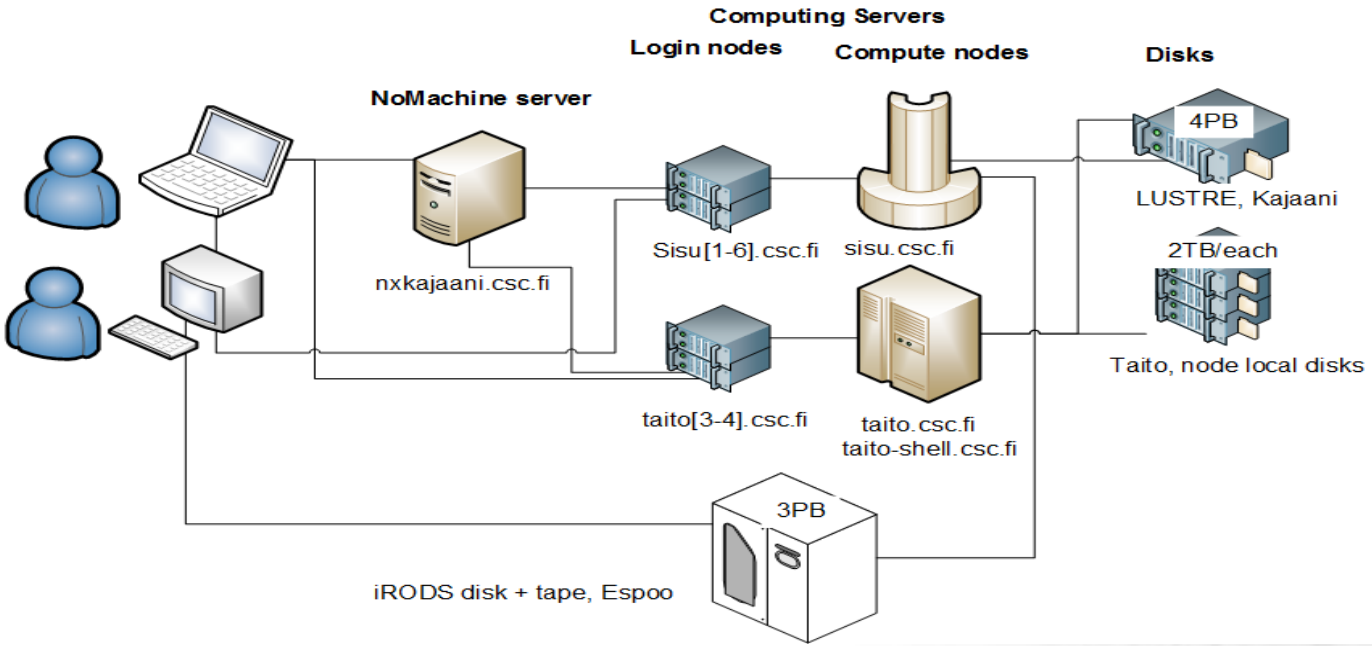
- HAKA is the identity federation of the Finnish universities, polytechnics and research institutions.
- 280000 users
- HAKA authentication gives access with your university account and password to:
  - SUI
  - Eduroam
  - …

Connecting to CSC

Learning targets

- Be aware of different ways of accessing CSC resources
- Logged in to Taito with ssh and NoMachine

The (almost) Complete Picture



Access via any of:

- Ssh
- NoMachine
- Browser (SUI)
- Tunneling
- ARC (FGCI)
- HAKA
- iRODS

Computing servers

Sisu: Cray XC40

- 1688 x 24 Intel 2.6 GHz = 40512 cores
- 2.7 GB mem / core
- Aries interconnect
- Massively parallel jobs
- Only batch jobs

Taito: HP ProLiant SL 230s + Apollo 6000 XL230a G9

- 576 x 16 Intel Sandy Bridge 2.6 GHz = 9216 cores
  - 4/16 GB memory / core (64/256 GB / node)
- 407 x 24 Intel Haswell 2.6 GHz = 9768 cores
  - 5.3/10.6 GB memory / core (128/256 GB / node)
- 2 x 32 + 4 x 40 = 224 cores (Hugemem)
  - 48 GB/core (1.5 TB/node)
- FDR Infiniband
- Serial and parallel jobs
- Very large memory jobs
- Interactive jobs (taito-shell)
- Cloud servers also on this hardware

Direct ssh connection –Linux/Mac

- From UNIX/Linux/OSX command line
- Use –X (or –Y) to enable remote graphics*
- scp : copy file to remote machine

```
$ ssh -X yourid@taito.csc.fi
```

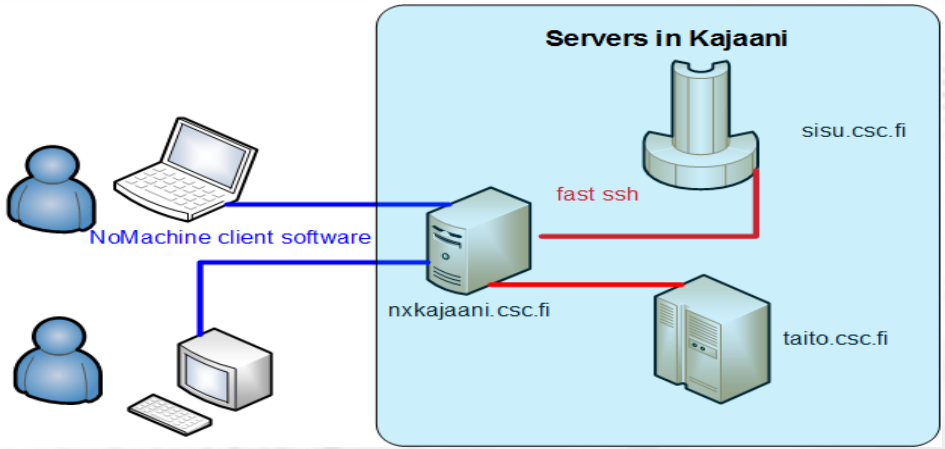
```
$ scp file yourid@taito.csc.fi:
```

```
login as: yourid
Last login: Tue Jul 09 13:14:15 2019 from cool.csc.fi
┌─ Welcome ──────────────────────────────────────────────────────────────────┐
│         CSC - Tieteen tietotekniikan keskus - IT Center for Science        │
│                  HP Cluster Platform SL230s Gen8 TAITO                     │
├─ Contact ──────────────────────────────────────────────────────────────────┤
...
```

* In Windows you'd also need an X-windows emulator, but there is a better way

NoMachine Remote Desktop

- Client connection between user and gateway
- Good performance even with slow network
- Ssh from gateway to server (fast if local)
- Persistent connection
- Suspendable
  - Continue later at another location
- Read the instructions…
  - ssh-key, keyboard layout, mac specific workarounds, …
- Choose an application or server to use (right click)



Access with scientific software

- Some software can be configured to use CSC servers directly, e.g.
  - TMolex, ADF, Maestro, Discovery Studio, Matlab
- The GUIs can be used to create and submit jobs directly to the Taito queueing system

Finnish Grid and Cloud Infrastructure - FGCI

- Distributed computing capacity
- 9 universities + CSC
- Requires a certificate
- Lots of preinstalled software
- Access with ARC –client
- From your own computer or Taito


```
arcproxy
arcsub jobscript.xrsl
arcget gsiftp://usva.fgi.csc.fi:2811/jobs/1246513389098765 4
```

- FGCI guide

Pouta Cloud service

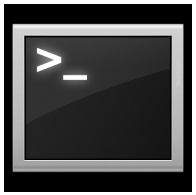
Do I need…

Different operating system and software stack than CSC's systems?

To run web services?

To extend my local computing resources?

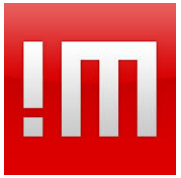
→ http://research.csc.fi/cloud-computing



Ascii terminal

- Open a terminal on your workstation (right click on backround or select from menu), then in terminal:

```
$ ssh user@taito.csc.fi
  (man in the middle?)
$ ls
$ hostname
$ gnuplot
$ plot sin(x)
```



NoMachine

- Open *NoMachine* client
- Select nxkajaani.csc.fi
- Insert your *username* and *password*
- (accept help screens)
- Right click on the background, choose taito from menu
- Give your password

```
$ ls
$ hostname
$ …
```

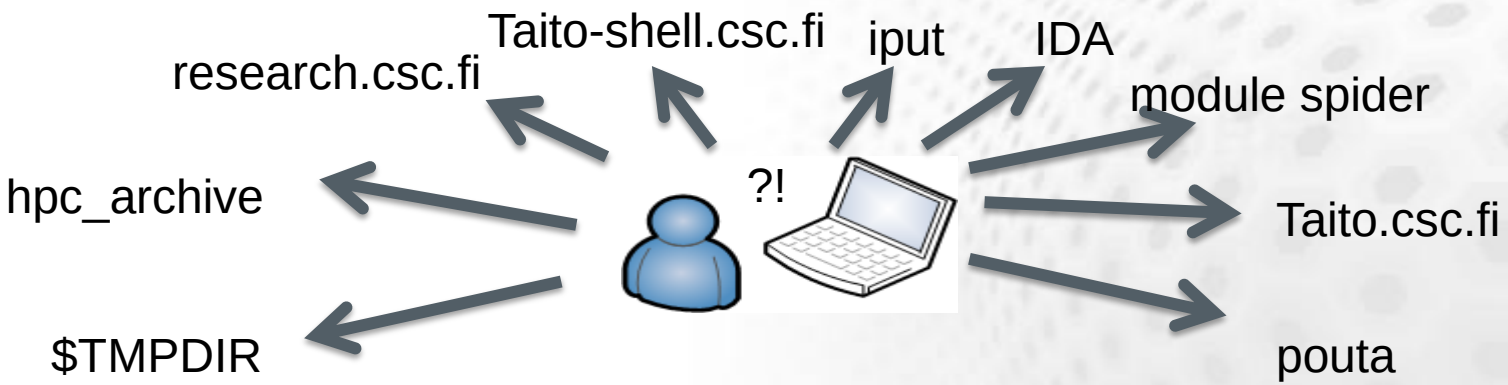
Summary: How to access resources at CSC

- Ssh terminal connection to CSC (+ X-term emulator for win)
- Installation at your own computer, license from CSC
  - Materials Studio, Discovery Studio, Ansys, …
- GUI at your own computer, computation at CSC (ssh pipe)
  - Tmolex, ADFgui, Discovery Studio
- GUI at your own computer, input files to CSC by hand, jobs launched from command prompt
- Scientist's User Interface (www based) sui.csc.fi
  - File manager, certificates, terminal, software distribution, …
- ARC (Nordugrid) middleware to run jobs in FGCI
- NoMachine Remote desktop (etätyöpöytä)
  - Client installed at your own computer, working with graphics at CSC
- Cloud services: pouta.csc.fi
  - Lots of freedom/flexibility and hence some administration and configuration work

CSC Computing Environment

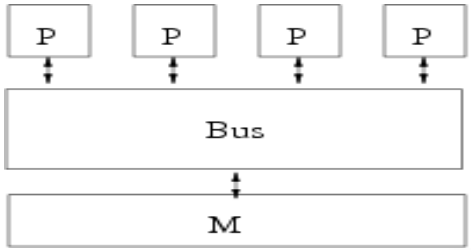
Learning target

- Know how to choose right server (resource)
- Know where to put your files
- Know how to setup and use preinstalled software

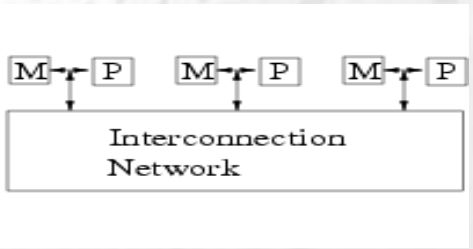


On Clusters and Supercomputers (1/2)

- Shared Memory Parallel (SMP):
  - All processors access (more or less) the same memory
  - Within node



- Distributed Memory:
  - Processes access their own memory
  - Interconnection network for exchange
  - Between nodes



On Clusters and Supercomputers (2/2)

- A cluster is a connection of separate units (nodes) via a fast network
- All larger CSC platforms (Sisu, Taito, FGCI) are clusters in a general sense

Server use profiles

- Taito (HP)
- Serial and parallel upto 448/672 cores
- Huge memory jobs
- Lots of preinstalled software

- Taito-shell (HP)
- Interactive jobs
- Very long jobs
- Auto queue, shared resources

- Sisu (Cray XE40)
- Parallel from 72 up to thousands of cores
- Scaling tests 1008+

- cPouta (HP) Cloud
- Serial and parallel upto 16 cores

- FGCI (Dell/HP)
- Serial and parallel (16)

Main Computing capacity: Sisu,Taito FGCI

	Sisu (Phase 2)	Taito (Phase 2)	FGCI prerelease
Availability	2014-	2015-	2016-
CPU	Intel Haswell and Sandy Bridge, 2 x 12 and 2 x 8 cores, 2.6 GHz, Xeon E5-2690v3 and E5-2670		Intel Xeon, 2 x 6 cores, 2.7 GHZ, X5650 and 4x12 Intel Xeon CPU E7-4830v3 @2.1GHz
Interconnect	Aries	FDR IB	QDR IB
Cores	40512	9768+9216	7308+3600
RAM/node	64 GB	64/128/256/ 1536 GB	128/256/512 GB
Tflops	1688	515	218
GPU nodes	-	50	8
Disc space	4 PB	4 PB	1+ PB

FGCI – The Finnish Grid and Cloud Infrastructure

- Consortium of 9 Finnish Universities and CSC
- Infrastructure consists of 7368+3600 cores and 100 GPU cards (+ Taito)
- Accessed via ARC middleware
- Submit jobs from taito/own workstation
- Preinstalled software
- More information: FGCI guide

FINNISH GRID AND CLOUD INFRASTRUCTURE

Sample ARC job description file

```
&
(executable=runbwa.sh)
(jobname=bwa_1)
(stdout=std.out)
(stderr=std.err)
(gmlog=gridlog_1)
(walltime=24h)
(memory=8000)
(disk=4000)
(runtimeenvironment>="APPS/BIO/BWA_0.6.1")
(inputfiles=
( "query.fastq" "query.fastq" )
( "genome.fa" "genome.fa" )
)
(outputfiles=
  ( "output.sam" "output.sam" )
)
```

IaaS cloud services

https://research.csc.fi/cloud-computing

- Infrastructure as a Service (IaaS) type of cloud
- OpenStack cloud middleware for management
- The Virtual Machines are admistrated by the user

- **cPouta**
  - The cPouta service allows customers to run virtual machines connected to the Internet.
  - PI of a project can apply for access in SUI
  - Youtube videos on how to start a VM in cPouta

- **ePouta**
  - The cloud service combines virtual computational resources with the customers' own resources using a dedicated light path or MPLS connection.
  - Designed for secure data handling

Directories at CSC Environment (1)

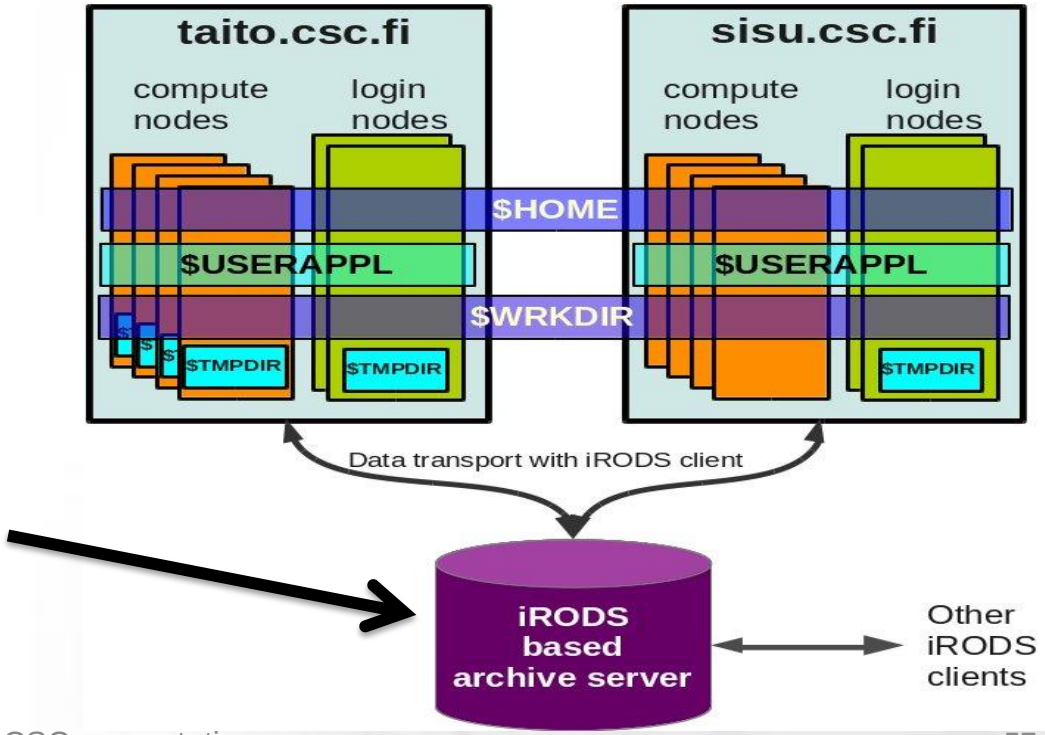
https://research.csc.fi/data-environment

Directory or storage area	Intended use	Default quota/user	Storage time	Backup
$HOME [1]	Initialization scripts, source codes, small data files. Not for running programs or research data.	50 GB	Permanent	Yes
$USERAPPL [1]	Users' own application software.	50 GB	Permanent	Yes
$WRKDIR [1]	Temporary data storage.	5 TB	Until further notice.	No
$TMPDIR [3]	Temporary users' files.	-	~2 days	No
Project [1]	Common storage for project members. A project can consist of one or more user accounts.	On request.	Permanent	No
HPC Archive [2]	Long term storage.	2 TB	Permanent	Yes
IDA [2]	Sharing and long term storage	several TB	At least -2017	Yes

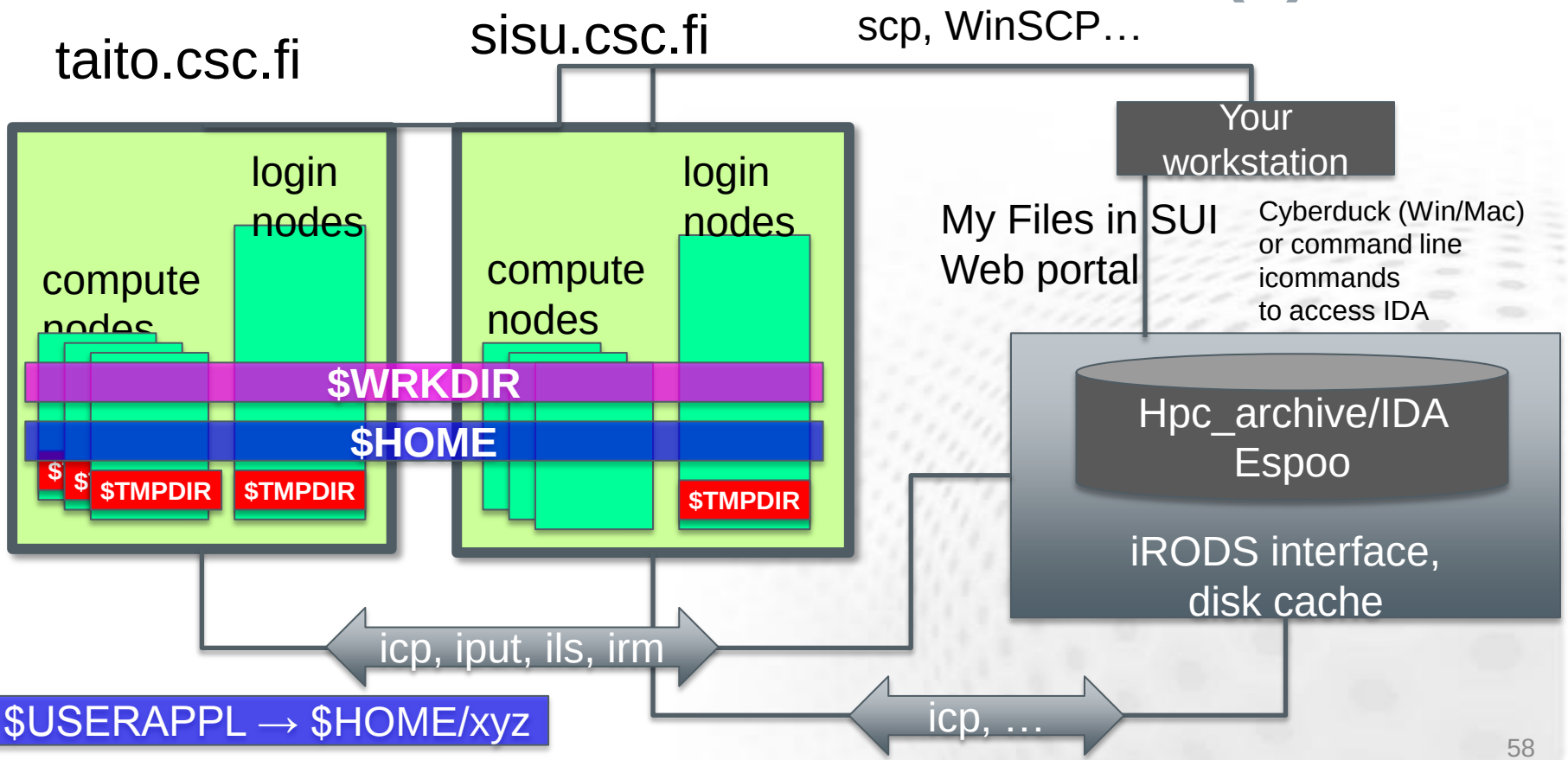
[1]: Lustre parallel ([3]:local) file system in Kajaani [2]: iRODS storage system in Espoo

Directories at CSC Environment (2)

- What can be seen from where
- Use **$TMPDIR** for fast/random file i/o *e.g.* compiling
- IDA/hpc_archive accessed with i-commands
- http://openscience.fi/ida-user-instructions



Directories at CSC Environment (3)



Storage: hard disks

- 4 PB on DDN (Lustre), Sisu and Taito
  - **$USERAPPL**: *put your own applications here*
  - /homeappl/home/username/app_taito
  - /homeappl/home/username/app_sisu
  - ***/tmp*** (Taito, ~2 TB) to be used for *e.g. compiling codes on the login nodes*
  - **$TMPDIR** on compute nodes: *for scratch files (accessed with $TMPDIR in batch script)*
  - **$HOME** for configuration files and misc. Smallish storage. If full, gives strange errors (X-graphics etc.)
  - **$WRKDIR** for large data and during calculations. Avoid lots of small files.

Storage: disks and tape

- Disk/Tape space through IDA
  - Requires an application http://openscience.fi/becoming-an-ida-user
  - Resource allocation is assigned to universities, universities of applied sciences and Academy of Finland according to division based on Ministry indicators
  - Storage space usage via browser, command line or file transfer program
  - http://openscience.fi/ida-user-instructions
  - Flexible sharing with colleagues/collaborators/public
- Tape (+ disk cache) as **hpc_archive**
  - Default long term storage
  - Access with i-commands from Sisu/Taito

IDA/hpc_archive interfaces at CSC

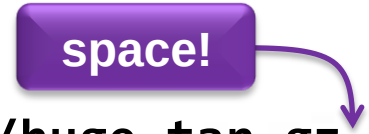
Some iRODS commands

- **`iput` *`file`*** move file to IDA
- **`iget` *`file`*** retrieve file from IDA
- **`ils`** list the current IDA directory
- **`icd` *`dir`*** change the IDA directory
- **`irm` *`file`*** remove file from IDA
- **`imv` *`file file`*** move file inside IDA
- **`irsync`** synchronize the local copy with the copy in IDA
- **`imkdir`** create a directory to IDA
- **`iinit`** Initialize your IDA account

Moving files, best practices

- rsync, not scp (when lots of/big files), *tar & zip first*

```
$ rsync -P username@taito-login3.csc.fi:/tmp/huge.tar.gz .
```



- Funet FileSender (max 50 GB [50GB as an attachment? No!])
  - *https://filesender.funet.fi*
  - Files can be downloaded also with **`wget`**
- iRODS, batch-like process, staging
- IDA: http://openscience.fi/ida

- CSC can help to tune e.g. TCP/IP parameters
- FUNET backbone 100 Gbit/s

https://research.csc.fi/csc-guide-moving-data-between-csc-and-local-environment

The module system

- Tool to set up your environment
  - Load libraries, adjust path, set environment variables
  - Needed on a server with hundreds of applications and several compilers etc.
- Slightly different on Taito vs. other systems
- Used both in interactive and batch jobs

Typical module commands

`module avail`	shows available modules (compatible modules in taito)
`module spider`	shows all available modules in taito
`module list`	shows currently loaded modules
`module load <name>`	loads module <name> (default version)
`module load <name/version>`	loads module <name/version>
`module switch <name1> <name2>`	unloads module name1 and loads module name2
`module purge`	unloads all loaded modules

Taito has ”meta-modules” named *e.g.* gromacs-env, which will load all necessary modules needed to run gromacs.

Module example

- Show compatible modules on Taito
  - `$ module avail`
- Initialize R and RStudio statistics packages
  - `$ module load r-env`
  - `$ module load rstudio`
- Start RStudio using the command
  - `$ rstudio`
- It's better to run the GUI (and calculations) on a compute node (jobs that have used 1h of CPU on the login node will be killed automatically)
- For interactive work, use taito-shell.csc.fi

Simple plotting in R

```
> a=seq(0,10,by=0.1)
> plot(a,cos(a))
```

Learning targets achieved?

- How to choose right server (resource)?
- Where to put your files?
- How to setup and use preinstalled software/libraries/compilers?

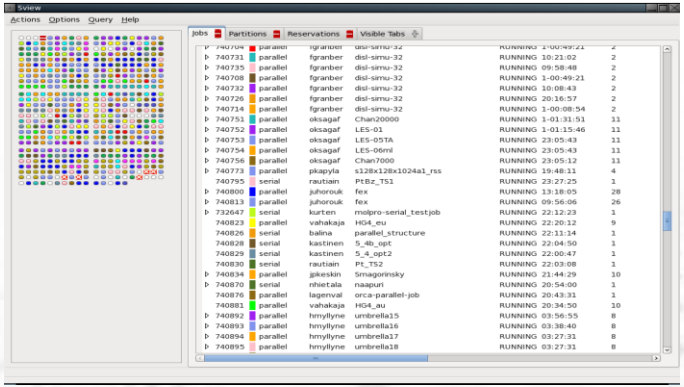
Running jobs at CSC

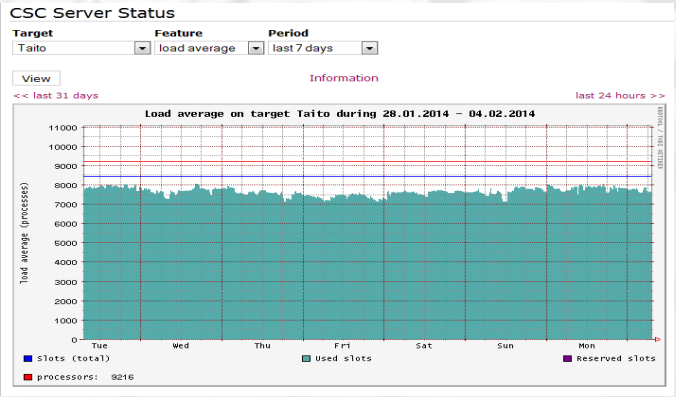
Batch jobs learning target

- Benefits of batch jobs for compute intensive jobs
  - Difference of login and compute node
- How to submit and monitor jobs
- Batch script contents *i.e.* resource requirements
- How to learn resource requirements of own jobs
- Be aware of batch script wizard in SUI
- Submit first job(s)
- Learn to read the the manual

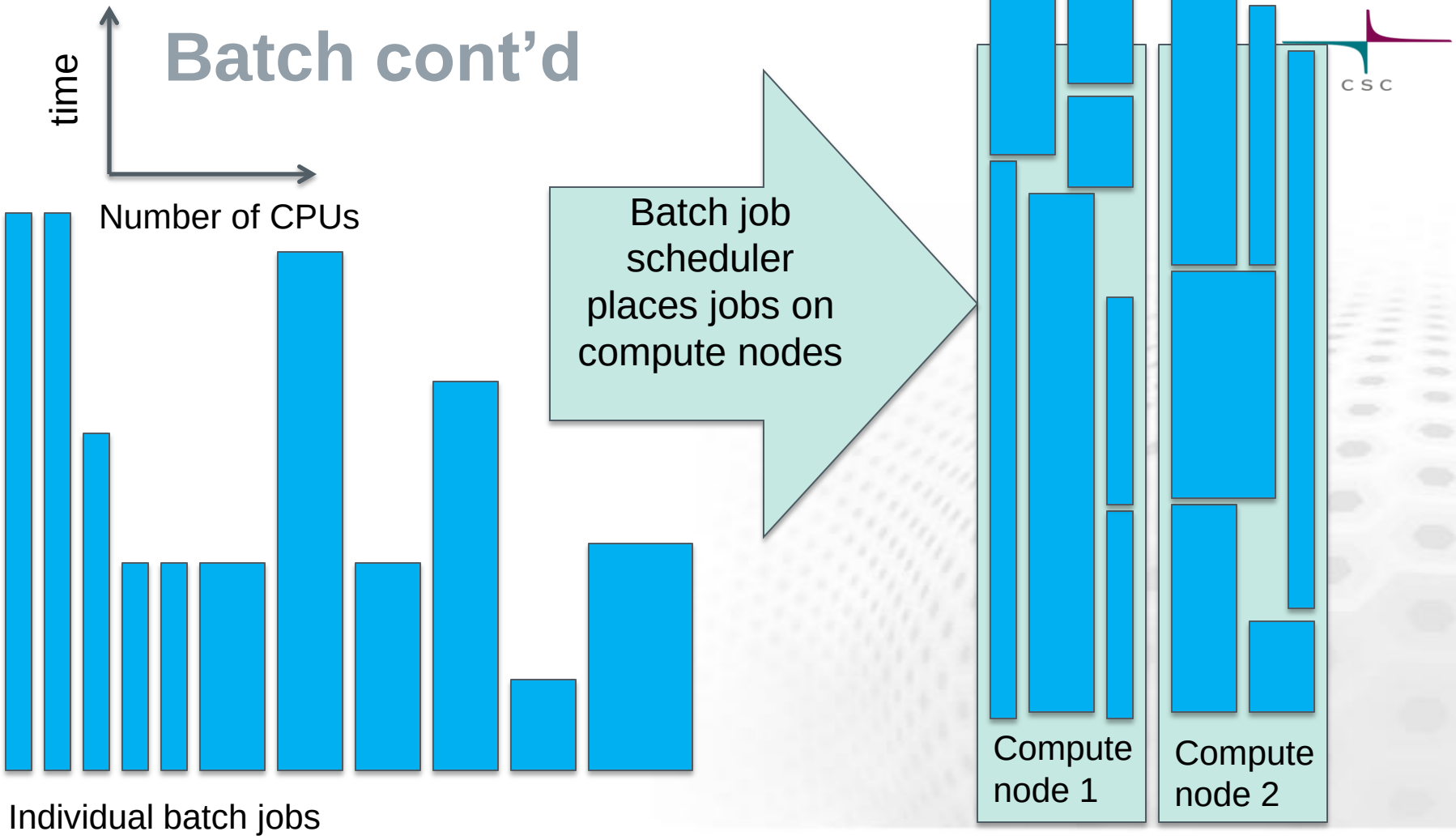
What is a batch system?

- Optimizes resource usage by filling the server with jobs
- Cores, memory, disk, length, …
- Jobs to run are chosen based on their priority
- Priority increases with queuing time
- Priority decreases with recently used resources
- Short jobs with little memory and cores queue the least
- CSC uses SLURM (Simple Linux Utility for Resource Management)





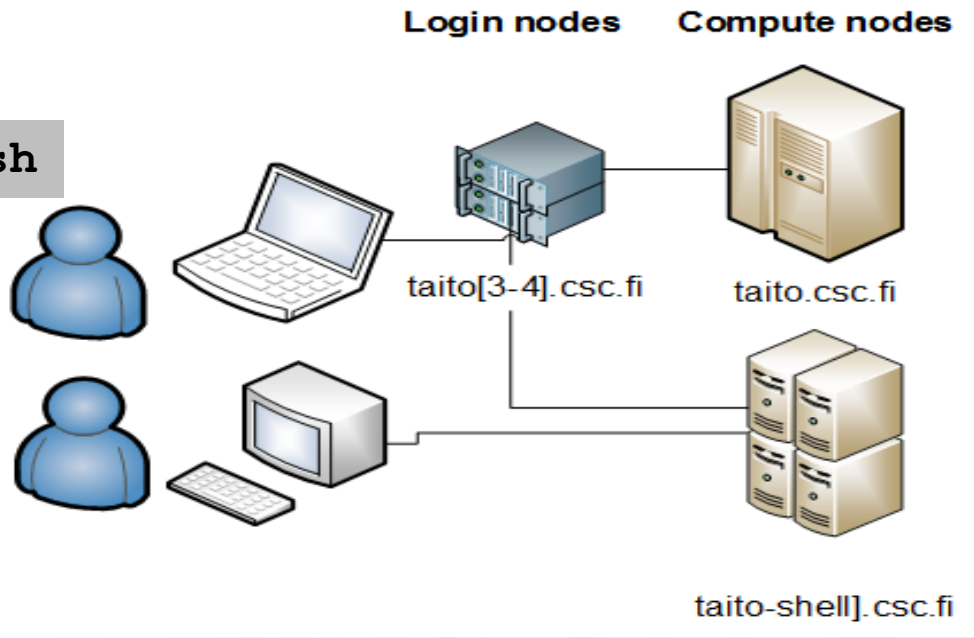
Batch cont'd


time
Number of CPUs
Individual batch jobs
Batch job scheduler places jobs on compute nodes
Compute node 1
Compute node 2
CSC

Compute nodes are used via queuing system

```
$ sbatch job_script.sh
```

```
$ ./my_prog &
```



Batch job overview

- Steps for running a batch job

1. Write a batch job script
   - Script details depend on server, check CSC Guides or software page!
   - You can use the Batch Job Script Wizard in Scientist's User Interface: https://sui.csc.fi/group/sui/batch-job-script-wizard

2. Make sure all the necessary files are in $WRKDIR
   - $HOME has limited space
   - **Login node** $TMPDIR is not available on compute nodes

3. Submit your job

```
$ sbatch myscript
```

Batch Job Script wizard in Scientist's User Interface

SUI / Services / Batch Job Script Wizard /

Batch Job Script Wizard

Host: taito

Application: Select application...

Level:

Form

General

Job Name: humppaa

Shell: /bin/bash

Email Address: atte.sillanpaa@csc.fi

Output

Standard Output File Name: ulos

Standard Error File Name: virheet

Computing Resources

Computing Time: 09:00:00

Number of Cores: 1

Memory Size: 2000

Memory request type: Per core

CPU architecture: Sandy Bridge or Haswell

Script Commands

```
# example run commands
srun ./my_serial_program
```

Script Result

```
#!/bin/bash -l
# created: Sep 6, 2016 10:26 AM
# author: asillanp
#SBATCH -J humppaa
#SBATCH --constraint="snb|hsw"
#SBATCH -o ulos
#SBATCH -e virheet
#SBATCH -p serial
#SBATCH -n 1
#SBATCH -t 09:00:00
#SBATCH --mem-per-cpu=2000
#SBATCH --mail-type=END
#SBATCH --mail-user=atte.sillanpaa@csc.fi

# commands to manage the batch script
#   submission command
#     sbatch [script-file]
#   status command
#     squeue -u asillanp
#   termination command
#     scancel [jobid]

# For more information
#   man sbatch
#   more examples in Taito guide in
#   http://research.csc.fi/taito-user-guide

# example run commands
srun ./my_serial_program

# This script will print some usage statistics to the
# end of file: ulos
# Use that to improve your resource request estimate
# on later jobs.
used_slurm_resources.bash
```

Save

Batch jobs: what and why

- User has to specify necessary resources
  - Can be added to the batch job script or given as command line options for sbatch (or a combination of script and command line options)
- Resources need to be adequate for the job
  - Too small memory reservation will cause the job to fail
  - When the time reservation ends, the job will be terminated whether finished or not
- But: Requested resources can affect the time the job spends in the queue
  - Especially number of cores and memory reservation
  - Using more cores does not always make the job run faster
  - Don’t request extra ”just in case” (time is less critical than memory wrt this)
- So: Realistic resource requests give best results
  - Not always easy to know beforehand
  - Usually best to try with smaller tasks first and check the used resources
  - You can check what was actually used with the **`sacct`** command

SLURM batch script contents

Example serial batch job script on Taito

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial
#SBATCH --constraint=snb

module load myprog
srun myprog -option1 -option2
```

#!/bin/bash -l

- Tells the computer this is a script that should be run using bash shell
- Everything starting with ”**#SBATCH**” is passed on to the batch job system (Slurm)
- Everything (else) starting with ”**#** ” is considered a comment
- Everything else is executed as a command

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

#SBATCH -J myjob

- Sets the name of the job
- When listing jobs *e.g.* with **squeue**, only 8 first characters of job name are displayed.

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

```
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
```

- Option **`-e`** sets the name of the file where possible error messages (stderr) are written
- Option **`-o`** sets the name of the file where the standard output (stdout) is written
- When running the program interactively these would be written to the command promt
- What gets written to stderr and stderr depends on the program. If you are unfamiliar with the program, it's always safest to capture both
- **`%j`** is replaced with the job id number in the actual file name

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

```
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
```

- Option **`--mail-type=END`** = send email when the job finishes
- Option **`--mail-user`** = your email address.
- If these are selected you get a email message when the job is done. This message also has a resource usage summary that can help in setting batch script parameters in the future.
- To see actually used resources try also: **`sacct -l -j <jobid>`** (more on this later)

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

`#SBATCH --mem-per-cpu=4000`

- The amount of memory reserved for the job in MB
  - 1000 MB = 1 GB
- Memory is reserved per-core basis even for shared memory (OpenMP) jobs
  - For those jobs it is better to ask memory *per job*:
  - **`--mem=1000`**
- Keep in mind the specifications for the nodes. Jobs with impossible requests are rejected (try **`squeue`** after submit)
- If you reserve too little memory the job will be killed (you will see a corresponding error in the output)
- If you reserve too much memory your job will spend much longer in queue and potentially waste resources (idle cores)

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

```
#SBATCH -t 02:00:00
```

TIP: If you're unsure of the syntax, use Batch job wizard in SUI

- Time reserved for the job in hh:mm:ss
- When the time runs out the job will be terminated!
- With longer reservations the job may queue longer
- Limit for normal serial jobs is 3d (72 h)
  - if you reserve longer time, choose "longrun" queue (limit 14d)
  - In the longrun queue you run at your own risk. If a batch job in that queue stops prematurely no compensation is given for lost cpu time
  - In longrun you likely queue for a longer time: shorter jobs and restarts are better (safer, more efficient)
- Default job length is 5 minutes → need to be set by yourself.

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

#SBATCH -n 1

- Number of cores to use. More than one means parallel.
- It's also possible to control on how many nodes your job is distributed. Normally, this is not needed. By default use all cores in allocated nodes:
  - **--ntasks-per-node=16 #(Sandy Bridge)**
  - **--ntasks-per-node=24 #(Haswell)**
- Check documentation: http://research.csc.fi/software
  - There's a lot of software that can only be run in serial
- OpenMP applications can only use cores in one node

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

#SBATCH -p serial

- The queue the job should be submitted to
- Queues are called ”partitions” in SLURM
- You can check the available queues with command

  sinfo -l

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

```
[asillanp@taito-login4 ~]$ sinfo -l
Wed Jan 28 15:45:39 2015
PARTITION AVAIL  TIMELIMIT   JOB_SIZE ROOT    SHARE      GROUPS  NODES       STATE NODELIST
serial*      up 3-00:00:00          1   no       NO         all      1    draining c623
serial*      up 3-00:00:00          1   no       NO         all    101       mixed c[25,76-77,…
serial*      up 3-00:00:00          1   no       NO         all    593   allocated c[3-24,26-75,…
serial*      up 3-00:00:00          1   no       NO         all    226        idle c[211-213,…
parallel     up 3-00:00:00       1-28   no       NO         all      1    draining c623
parallel     up 3-00:00:00       1-28   no       NO         all    101       mixed c[25,76-77,…
parallel     up 3-00:00:00       1-28   no       NO         all    593   allocated c[3-24,26-75,…
parallel     up 3-00:00:00       1-28   no       NO         all    226        idle c[211-213,…
longrun      up 14-00:00:0          1   no       NO         all      1    draining c623
longrun      up 14-00:00:0          1   no       NO         all    101       mixed c[25,76-77,…
longrun      up 14-00:00:0          1   no       NO         all    587   allocated c[3-24,26-75,…
longrun      up 14-00:00:0          1   no       NO         all    226        idle c[211-213,…
test         up      30:00        1-2   no       NO         all      4        idle c[1-2,984-985]
hugemem      up 7-00:00:00          1   no       NO         all      2       mixed c[577-578]
```

#SBATCH --constraint=snb

- The job is run only in Sandy Bridge (snb) nodes
- The other option is Haswell node (hsw) or
  - **#SBATCH --constraint=hsw**
- Either that is free ”snb|hsw”
  - **#SBATCH --constraint=”snb|hsw”**

- Currently the default is to use *either* architecture in *serial* and *longrun* partitions
- Sandy Bridge in *test* and *parallel*
- A single job cannot use CPUs from both architectures, but SLURM will take care of this

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial
#SBATCH --constraint=snb

module load myprog
srun myprog -option1 -option2
```

```
module load myprog
srun myprog -option1 -option2
```

- Your commands
  - These define the actual job to performed: these commands are run on the compute node.
  - See application documentation for correct syntax
  - Some examples also from batch script wizard in SUI
- Remember to load modules if necessary
- By default the working directory is the directory where you submitted the job
  - If you include a **`cd`** command, make sure it points to correct directory
- Remember that input and output files should be in $WRKDIR (or in some case $TMPDIR)
- $TMPDIR contents are deleted after the job
- **`srun`** tells your program which cores to use. There are also exceptions…

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

Most commonly used sbatch options

Slurm option	Description
`--begin=time`	defer job until HH:MM MM/DD/YY
`-c, --cpus-per-task=ncpus`	number of cpus required per task
`-d, --dependency=type:jobid`	defer job until condition on jobid is satisfied
`-e, --error=err`	file for batch script's standard error
`--ntasks-per-node=n`	number of tasks per node
`-J, --job-name=jobname`	name of job
`--mail-type=type`	notify on state change: BEGIN, END, FAIL or ALL
`--mail-user=user`	who to send email notification for job state changes
`-n, --ntasks=ntasks`	number of tasks to run
`-N, --nodes=N`	number of nodes on which to run
`-o, --output=out`	file for batch script's standard output
`-t, --time=minutes`	time limit in format hh:mm:ss
`--mem-per-cpu=<number in MB>`	maximum amount of real memory per allocated cpu required by the job in megabytes
`--mem=<number in MB>`	maximum memory per node

SLURM:
Managing batch jobs in Taito

Submitting and cancelling jobs

- The script file is submitted with command

```
$ sbatch batch_job.file
```

- Job can be deleted with command

```
$ scancel <jobid>
```

Queues

- The job can be followed with command **`squeue`**:

```
$ squeue                          (shows all jobs in all queues)
$ squeue -p <partition>           (shows all jobs in single queue (partition))
$ squeue -u <username>            (shows all jobs for a single user)
$ squeue -j <jobid> -l            (status of a single job in long format)
```

- To estimate the start time of a job in queue

```
$ scontrol show job <jobid>
```

row "StartTime=..." gives an *estimate* on the job start-up time, e.g.

```
StartTime=2014-02-11T19:46:44 EndTime=Unknown
```

- **`scontrol`** will also show where your job is running
- If you add this to the end of your batch script, you'll get additional info to stdout about resource usage (works for jobs run with srun)
  - **`used_slurm_resources.bash`**

Job logs

- Command **`sacct`** can be used to study *past* jobs
  - Useful when deciding proper resource requests

TIP: Check MaxRSS to see how much memory you need and avoid overbooking

`$ sacct`	Short format listing of jobs starting from midnight today
`$ sacct -l`	long format output
`$ sacct -j <jobid>`	information on single job
`$ sacct -S YYYY-MM-DD`	listing start date
`$ sacct -o`	list only named data fields, e.g.
`$ sacct -u <username>`	list only jobs submitted by username

```
$ sacct -o jobid,jobname,maxrss,reqmem,elapsed -j <jobid>
```

Available nodes/queues and limits

- You can check available resources per node in each queue:
  `$ sjstat -c`

```
-----------------------------------------------------------------
Pool         Memory  Cpus  Total Usable   Free  Other Traits
-----------------------------------------------------------------
serial*    258000Mb    24     10     10      5  hsw,haswell
serial*     64300Mb    16    502    502      9  snb,sandybridge
serial*    258000Mb    16     14     14      0  bigmem,snb,sandybridge
serial*    128600Mb    24    395    395      6  hsw,haswell
parallel   258000Mb    24     10     10      5  hsw,haswell
parallel    64300Mb    16    502    502      9  snb,sandybridge
parallel   258000Mb    16     14     14      0  bigmem,snb,sandybridge
parallel   128600Mb    24    395    395      6  hsw,haswell
longrun    258000Mb    16      8      8      0  bigmem,snb,sandybridge
longrun    258000Mb    24     10     10      5  hsw,haswell
longrun     64300Mb    16    502    502      9  snb,sandybridge
longrun    128600Mb    24    395    395      6  hsw,haswell
test        64300Mb    16      2      2      2  snb,sandybridge
test       128600Mb    24      2      2      2  hsw,haswell
hugemem   1551000Mb    32      2      2      0  bigmem,snb,sandybridge
hugemem   1551000Mb    40      4      4      1  bigmem,hsw,haswell,ssd
```

Most frequently used SLURM commands

Command	Description
`srun`	Run a parallel job.
`salloc`	Allocate resources for interactive use.
`sbatch`	Submit a job script to a queue.
`scancel`	Cancel jobs or job steps.
`sinfo`	View information about SLURM nodes and partitions.
`squeue`	View information about jobs located in the SLURM scheduling queue
`smap`	Graphically view information about SLURM jobs, partitions, and set configurations parameters
`sjstat`	display statistics of jobs under control of SLURM (combines data from sinfo, squeue and scontrol)
`scontrol`	View SLURM configuration and state.
`sacct`	Displays accounting data for batch jobs.

Parallel jobs (1/2)

- Only applicable if your program supports parallel running
- Check application documentation for number of cores to use
  - Speed-up is often not linear (communication overhead)
  - Maximum number can be limited by the algorithms
  - Make sure (test) that using more cores speeds up calculation
- Mainly two types: MPI jobs and shared memory (OpenMP) jobs
  - OpenMP jobs can be run only inside one node
    - All cores access same memory space
  - MPI jobs can span several nodes
    - Each core has its own memory space
  - In some cases you can use both: MPI between nodes and OpenMP within a node. Check the documentation of your program

Parallel jobs (2/2)

- Memory can be reserved either per core or per node
  - For OpenMP jobs request memory per node (--mem=NN)
  - Don’t overallocate memory
  - If you reserve a complete node, you can also ask for all the memory
- Each server has different configuration so setting up parallel jobs in optimal way requires some thought
- See server guides for specifics: http://research.csc.fi/guides
  - Use Taito for large memory jobs
  - Sisu for massively parallel jobs
  - Check also the software specific pages for examples and detailed information: http://research.csc.fi/software

Array jobs (advanced usage)

- Best suited for running the same analysis for large number of files
- **`#SBATCH --array=1-100`**
- Defines to run 100 jobs, where a variable `$SLURM_ARRAY_TASK_ID` gets each number (1,2,…100) in turn as its value. This is then used to launch the actual job (*e.g.*
- **`$ srun myprog input_$SLURM_ARRAY_TASK_ID > output_ $SLURM_ARRAY_TASK_ID`**)
- Thus this would run 100 jobs:

```
srun myprog input_1 > output_1
srun myprog input_2 > output_2
…
srun myprog input_100 > output_100
```

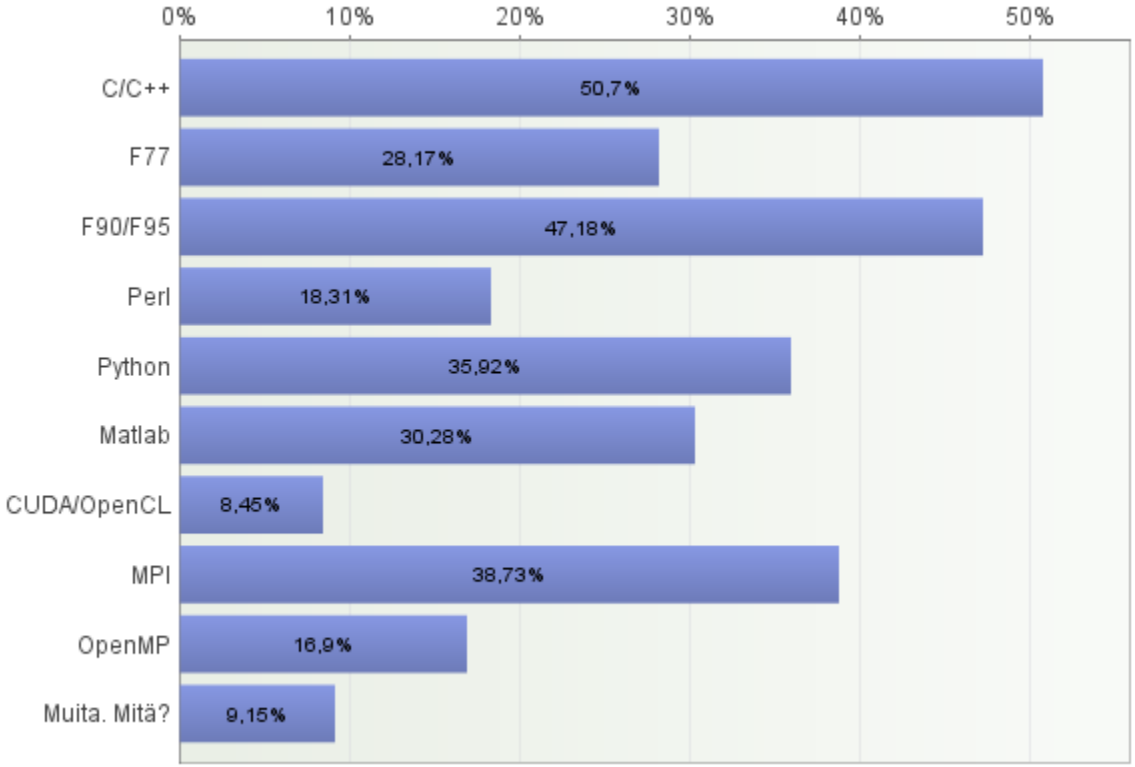
- For more information
  - http://research.csc.fi/taito-array-jobs

Compiling your program

What is a program?

- A *program* is a sequence of instructions understandable by a computer's central processing unit (CPU) that indicates which operations the computer should perform
  - Ready-to-run programs are stored as *executable* files
  - An executable file is a file that has been converted from source code into machine code, by a specialized program called a compiler

Programming languages at supercomputers



gcc [source files] [-o prog]

- Compiles C source files into a program
- *-o* to give the name of the program, defaults to a.out
- -c to compile into .o -files

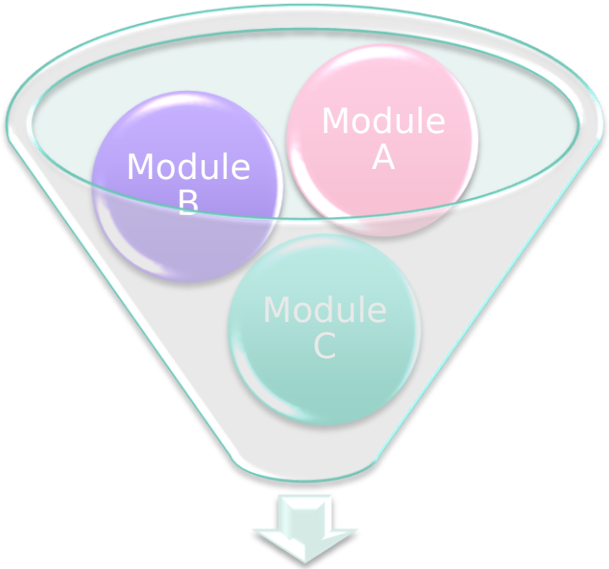
Compiling and installing programs

- For most programs, the three commands to compile and install in directory /home/user/programs are:

```
$ ./configure --prefix=/home/user/programs
$ make
$ make install
```

- **`make`** will be discussed in detail later today
- Destination for own programs in CSC computing environment: *$USERAPPL*

Why make?



- program separated into several files
- multiple inter-dependant modules
- compilation and linking becomes easily a nightmare
  - especially when developing the program!

Why make?

- when code has been modified, there are two approaches to compile the program:
  - re-compile everything → too slow
  - keep records and re-compile only what is needed → too much work
- **`make`** makes life easier by taking care of all the book keeping

Makefile

- defines:
  - work-flow(s) for producing target(s)
  - dependencies of each target
  - library paths, compiler flags etc.
- directives for conditional definitions etc.
- # starts a comment
- usually called `Makefile`
  - other choices: `makefile`, `GNUmakefile`

Basic syntax

RULE

name (usually filename)

list of files / rules

```
target: dependencies
    recipe
    ...
```

commands to execute

example:

```
foo.o: foo.c bar.h    # module foo
    cc -c foo.c

clean:                # remove all
    rm *.o
```

Note: use tabs instead of spaces to indent recipes!

Basic syntax

- *target*
  - usually the file that is produced by the recipe
  - name of an action also commonly used
    - for example: clean, distclean
- *dependencies*
  - a list of (source) files needed by the recipe
  - may also be other targets
- *recipe*
  - a list of commands to execute to make target

Logic of make

- read general macro definitions etc.
- call the rule for *target*
  - check when *dependencies* were changed
  - if any of the *dependencies* have changed, the *target* is re-built according to the *recipe*
- *dependencies* may also be *targets* for other rules
  - in that case, make calls those rules

Simple example

```
hello: main.o sub1.o sub2.o sub3.o
   f90 -o hello main.o sub1.o sub2.o sub3.o
main.o: main.f90
   f90 -c main.f90
sub1.o: sub1.f90
   f90 -c sub1.f90
sub2.o: sub2.f90
   f90 -c sub2.f90
sub3.o: sub3.f90
   f90 -c sub3.f90
clean:
   rm hello main.o sub1.o sub2.o sub3.o
```

Which target?

- by default, the first target is called
  - ’`hello`’ in the previous example
- target can be also specified when running `make`
  - `make target`
  - `make clean`
  - `make main.o`

Variables

- contain a string of text

```
variable = value
```

- substituted in-place when referenced

  $(variable) → value

- sometimes also called macros
- shell variables are also available in the makefile
  - $(HOME), $(USER), …

Two flavors of variables in GNU make

- recursive variables
  - defined as: `foo = bar`
  - expanded when referenced

- simple / constant variables
  - defined as: `foo := bar`
  - expanded when defined

```
foo = $(bar)
bar = $(ugh)
ugh = Huh?

$(foo) → Huh?

x := foo
y := $(x) bar
x = later

$(x) → later
$(y) → foo bar
```

Variables

- by convention variables are name in ALL-CAPS

- in the previous example we could have used a variable to store the names of all objects
  - `OBJ = main.o sub1.o sub2.o sub3.o`

Simple example revisited

```
OBJ = main.o sub1.o sub2.o sub3.o
hello: $(OBJ)
   f90 -o hello $(OBJ)
main.o: main.f90
   f90 -c main.f90
sub1.o: sub1.f90
   f90 -c sub1.f90
sub2.o: sub2.f90
   f90 -c sub2.f90
sub3.o: sub3.f90
   f90 -c sub3.f90
clean:
   rm hello $(OBJ)
```

Common variables

- some common variables
  - CC
  - CFLAGS
  - FC
  - FCFLAGS
  - LDFLAGS
  - OBJ
  - SRC

Special variables

- $@
  - name of the target

```
client: client.c
	$(CC) client.c -o $@
```

- $<
  - name of the first dependency

```
client: client.c
	$(CC) $< -o $@
```

Special variables

- $+
  - list of all dependencies
- $^
  - list of all dependencies (duplicates removed)
- $?
  - list of dependencies more recent than target

```
client: client.c
	$(CC) $+ -o $@
```

Special variables

- $*
  - common prefix shared by the target and the dependencies

```
client: client.c
	$(CC) -c -o $*.o $*.c
```

Special characters

- / continues a line
- # starts a comment
- @ executes a command quietly
  - by default, make echos all commands executed
  - this can be prevented by using @-sign at the beginning of the command

```
@echo ”quiet echo”
```

→ `quiet echo`

```
echo ”normal echo”
```

→ echo ”normal echo”
normal echo

Special characters

- if there is an error executing a command, make stops
  - this can be prevented by using a – sign at the beginning of a command

```
clean:
	-rm hello
	-rm $(OBJ)
```

Implicit rules

- one can use special characters to define an implicit rule
- e.g. quite often target and dependencies share the name (different extensions)
  - define an implicit rule compiling an object file from a Fortran 90 source code file

```
%.o: %.f90
	$(F90) $(FFLAGS) -c -o $@ $<
```

Example revisited again

```
OBJ = main.o sub1.o sub2.o sub3.o

# implicit rule for compiling f90 files
%.o: %.f90
  f90 -c -o $@ $<

hello: $(OBJ)
  f90 -o hello $(OBJ)

clean:
  rm hello $(OBJ)
```

Built-in functions

- GNU make has also built-in functions
  - for a complete list see:
    - `www.gnu.org/software/make/manual/make.html#Functions`
- strip, patsubst, sort, …
- dir, suffix, basename, wildcard, …
- general syntax
  - `$(function arguments)`

Command line options

- `-j` parallel execution
- `-n` dry-run
  - shows the command, but does not execute them
- `-p` print defaults
  - shows default rules and values for variables before execution
- `-s` silent-run
  - do not print commands as they are executed

Command line options

- variables can also be defined from the command line
  - `make CC=gcc "CFLAGS=-O3 -g" foobar`

Complete example

```
SRC = main.f90 sub1.f90 sub2.f90 sub3.f90
OBJ = $(patsubst %.f90, %.o, $(SRC))
F90 = gfortran
FFLAGS =
DEST = bin

# implicit rule for compiling f90 files
%.o: %.f90
   $(F90) $(FFLAGS) -c -o $@ $<

hello: $(DEST)/hello

$(DEST)/hello: $(OBJ)
   $(F90) $(FFLAGS) -o $@ $(OBJ)

clean:
   -rm $(OBJ)
   -rm $(DEST)/hello

# extra dependencies
sub2.o: modules.o
```

Science services at CSC: a short introduction

Software and databases at CSC

- Software selection at CSC:
  - http://research.csc.fi/software

Science discipline specific pages:

- http://research.csc.fi/biosciences
- http://research.csc.fi/chemistry

Chipster data analysis environment:

- http://chipster.csc.fi

Troubleshooter: Interactive session to deal with open questions and specific problems